Nos. 15-1470, -1554, -1556

IN THE
United States Court of Appeals for the Federal Circuit

MENTOR GRAPHICS CORPORATION, an Oregon corporation,
Plaintiff-Cross-Appellant,

v.

EVE-USA, INC., a Delaware corporation,
SYNOPSYS EMULATION AND VERIFICATION, S.A.S.,
formed under the laws of France,
SYNOPSYS, INC., a Delaware corporation,
Defendants-Appellants.

On Appeal from the U.S. District Court for the District of Oregon, Case Nos. 3:10-cv-00954-MO (Lead), 3:12-cv-01500-MO, 3:13-cv-00579-MO, Hon. Michael W. Mosman

OPENING BRIEF AND ADDENDUM OF DEFENDANTS-APPELLANTS EVE-USA, INC., SYNOPSYS EMULATION AND VERIFICATION, S.A.S., AND SYNOPSYS, INC.

I. Neel Chatterjee
Scott Lonardo
Travis Jensen
ORRICK, HERRINGTON & SUTCLIFFE LLP
1000 Marsh Road
Menlo Park, CA 94025

William H. Wright
ORRICK, HERRINGTON & SUTCLIFFE LLP
777 South Figueroa Street
Los Angeles, CA 90017

E. Joshua Rosenkranz
Andrew D. Silverman
Daniel A. Rubens
ORRICK, HERRINGTON & SUTCLIFFE LLP
51 West 52nd Street
New York, NY 10019
(212) 506-5000
jrosenkranz@orrick.com

Eric A. Shumsky
Robert M. Loeb
ORRICK, HERRINGTON & SUTCLIFFE LLP
Columbia Center
1152 15th Street NW
Washington, DC 20005

Counsel for Defendants-Appellants

CERTIFICATE OF INTEREST

Counsel for Appellants Synopsys, Inc., Synopsys Emulation and Verification S.A.S, and EVE-USA, Inc. certifies the following:

1. The full names of every party or amicus represented by me are: Synopsys, Inc., EVE-USA, Inc., and Synopsys Emulation and Verification S.A.S.

2. The name of the real party in interest (if the party named in the caption is not the real party in interest) represented by me: N/A.

3. All parent corporations and any publicly held companies that own 10% or more of the stock of the parties represented by me are:

(a) Synopsys, Inc. has no parent corporation.

(b) EVE-USA, Inc. is a wholly owned subsidiary of Synopsys, Inc., a publicly held corporation.

(c) Synopsys Emulation and Verification S.A.S. is owned by Synopsys Global Licensing and Distribution Limited Liability Company. Synopsys Global Licensing and Distribution Limited Liability Company is owned by Synopsys Ireland Limited. Synopsys Ireland Limited is owned by Synopsys, Inc., a publicly held corporation. No publicly held corporation directly owns 10% or more of Synopsys Emulation and Verification S.A.S.'s stock.

4. The names of all law firms and the partners or associates that appeared for parties or amicus now represented by me in the trial court or are expected to appear in this court are:

COOLEY LLP
Reuben H. Chen
Kathryn D. Duvall
Christen M.R. Dubois (no longer with firm)
Michael G. Rhodes
Ricardo Rodriguez
Javier Torres (no longer with firm)

KASOWITZ, BENSON, TORRES & FRIEDMAN LLP
Uttam Dubal (no longer with firm)
Gabriel S. Gross (no longer with firm)
Joseph H. Lee (no longer with firm)
Douglas E. Lumish (no longer with firm)
Jeffrey G. Homrig (no longer with firm)

MILLER NASH GRAHAM & DUNN LLP
Dennis P. Rawlinson
Justin C. Sawyer

ORRICK, HERRINGTON & SUTCLIFFE LLP
Robert J. Benson
James C. Brooks
I. Neel Chatterjee
Gino Cheng (no longer with firm)
Jesse Y. Cheng
Vickie L. Feeman
Travis Jensen
Scott D. Lindlaw (no longer with firm)

Robert M. Loeb
Scott Lonardo
Richard F. Martinelli
Christopher R. Ottenweller
Cam T. Phan
E. Joshua Rosenkranz
Daniel A. Rubens
Eric A. Shumsky
Andrew D. Silverman
Christina M. Von der Ahe
William H. Wright

PERKINS COIE LLP
Scott D. Eads
Stephen F. English
Julia E. Markley

SIDLEY AUSTIN LLP
David T. DeZern
Leonard J. Denbina (no longer with firm)
Aseem S. Gupta (no longer with firm)
David T. Pritikin
M. Patricia Thayer
Sue Wang (no longer with firm)
Philip W. Woo

Date: July 17, 2015

ORRICK, HERRINGTON & SUTCLIFFE LLP

/s/ E. Joshua Rosenkranz
E. Joshua Rosenkranz

Counsel for Defendants-Appellants

TABLE OF CONTENTS

TABLE OF AUTHORITIES

TABLE OF ABBREVIATIONS AND GLOSSARY

’376 patent	U.S. Patent No. 6,240,376, assigned to Mentor Graphics Corporation
’526 patent	U.S. Patent No. 7,069,526, assigned to Synopsys, Inc.
’109 patent or “Gregory”	U.S. Patent No. 6,132,109, assigned to Synopsys, Inc.
APA	Administrative Procedure Act, 5 U.S.C. §§ 701-706
Board	Patent Trial and Appeal Board
EDA	Electronic Design Automation
EMVR	Entire Market Value Rule
EVE	EVE-USA, Inc. and Emulation and Verification S.A. (now named Synopsys Emulation and Verification S.A.S.)
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
Mentor	Cross-Appellant Mentor Graphics Corporation
PTO	United States Patent & Trademark Office
RTL	Register Transfer Level
Synopsys	Collectively, Appellants Synopsys, Inc., EVE-USA, Inc., and Synopsys Emulation and Verification S.A.S.
ZeBu emulator	Collectively, Synopsys’ ZeBu-Server, ZeBu-Server 2, ZeBu-Server 3, and ZeBu-Blade emulators

STATEMENT OF RELATED CASES

This appeal arises from litigation in the District of Oregon between Synopsys, Inc., Synopsys Emulation and Verification S.A.S., and EVE-USA, Inc. (collectively, "Synopsys") and Mentor Graphics Corporation ("Mentor"). Mentor asserted five patents against Synopsys, including U.S. Patent No. 6,240,376 ("the '376 patent"), and Synopsys asserted two patents against Mentor: U.S. Patent No. 6,132,109 ("the '109 patent," sometimes known as "Gregory") and U.S. Patent No. 7,069,526 ("the '526 patent").

The following pending matters may affect or be affected by this Court's decision. *See* Fed. Cir. R. 47.5.

1. Shortly before Synopsys brought the declaratory judgment action against Mentor concerning the '376 patent, Synopsys petitioned for *inter partes* review of that patent. *Synopsys, Inc. v. Mentor Graphics Corp.*, IPR2012-00042 (Patent Trial and Appeal Board). The Patent Trial and Appeal Board ("the Board") found a reasonable likelihood that one or more claims of the '376 patent are invalid, and accordingly instituted *inter partes* review as to claims 1-9, 11, and 28-29. The Board ultimately cancelled claims 5, 8, and 9.

Synopsys appealed to this Court the Board's decision not to cancel claims 1 and 28 (two of the claims upon which the final judgment and injunction in this litigation were based) and the Board's failure to issue a final written decision that addresses the patentability of all of the claims Synopsys challenged (including claims 24, 26, and 27, the remaining claims at issue here). Mentor cross-appealed, arguing that the Board should not have instituted review and that the Board erred in denying Mentor's conditional motion to amend claims 5, 8, and 9. *Synopsys, Inc. v. Mentor Graphics Corp.*, Nos. 2014-1516, -1530 (Fed. Cir.). Briefing has concluded, and the case is ready for oral argument, which has not yet been scheduled.

2. After the Board instituted *inter partes* review, Mentor sued the Patent & Trademark Office ("PTO") in the Eastern District of Virginia under the Administrative Procedure Act ("APA"), challenging the Board's decision to institute *inter partes* review. *Mentor Graphics Corp. v. Lee*, No. 1:13-cv-00518 CMH-TCB (E.D. Va.). The district court granted motions to dismiss filed by the PTO and Synopsys (which had intervened). Mentor appealed to this Court. Its appeal has been stayed pending resolution of *Versata Development Group, Inc. v. Lee*, Fed. Cir. No. 2014-

1145. *See* Order, *Mentor Graphics Corp. v. Lee*, No. 2013-1669 (Fed. Cir. July 30, 2014), ECF No. 37.

3. Synopsys also filed an APA suit in the Eastern District of Virginia. In that suit, Synopsys challenges the Board's regulation, policy, and practice of failing to "issue a final written decision with respect to the patentability of any patent claim challenged by the petitioner," 35 U.S.C. § 318(a), as required by statute, and instead issuing final written decisions that address only certain challenged claims. The district court dismissed Synopsys' complaint for lack of jurisdiction. *See Synopsys, Inc. v. Lee*, No. 1:14-cv-00674-JCC-IDD (E.D. Va.). Synopsys has appealed, the briefing recently concluded, and the case is currently awaiting argument. *Synopsys, Inc. v. Lee*, No. 2015-1183 (Fed. Cir.).

4. Synopsys sued Mentor in the Northern District of California for infringement of four patents not at issue in this case. One of the accused products is Mentor's emulator. The court found three of Synopsys' patents invalid, and Synopsys has appealed the district court's decision to this Court. *Synopsys, Inc. v. Mentor Graphics Corp.*, No. 15-1599 (Fed. Cir.). Briefing in that appeal is ongoing at this time on essentially the same schedule as this case.

INTRODUCTION

Electronic circuits power the economy. Tiny integrated circuits, like microchips, are indispensable components of electronic equipment, from cell phones and computers to home appliances and cutting-edge medical devices. If the microchips don't work, the equipment won't work. This case involves technology for ensuring that the circuits work.

Synopsys has long played a leading role in helping major companies test their microchip designs to detect and fix flaws (or "bugs"). In 2012, Synopsys acquired EVE-USA, Inc. and Emulation and Verification S.A. (collectively, "EVE"), which specialized in emulators. An emulator is a sophisticated machine that constructs a working physical model of a chip design and tests it for bugs. Mentor sued Synopsys, claiming that the emulation technology Synopsys acquired infringes various Mentor patents—only one of which survived to trial. Synopsys asserted two patents of its own, which the district court invalidated. A jury found that Synopsys' technology infringed Mentor's patent.

The jury verdict was flawed in two respects that cry out for appellate correction. *First*, Synopsys' emulators do not infringe as a matter of law. Microchip designers begin by writing source code that describes the circuit's functions. Mentor's patent teaches one way for the emulator to tell the designer which lines of source code executed during a test—specifically, to "identify" or "indicate" those lines of code—so if there is a bug, the designer will know where in the code to look for it. The plain meaning of those very same words, this Court has held, requires providing the user the specific information desired, not raw data from which the user can deduce the desired information. But it is undisputed that Synopsys' accused devices do no such thing. When Synopsys' emulator indicates there is a bug, it does not tell the designer what code executed during the test or where to find the bug in the source code. The designer must perform additional work to trace the emulation results to a specific line of code that was tested. Yet, when Synopsys explained to the district court that Mentor's theory of infringement therefore could not be sustained, the court declined to entertain the argument, describing it as "essentially boil[ing] down to … claim construction" issues that "[y]ou can make … to the Federal Circuit." A42,611-12. So here we are.

Second, the district court confessed that it committed "an error in the trial," A42,600, with an erroneous damages instruction. An emulator is a sophisticated device with numerous components. Mentor accused just two features, out of thousands in the device, of infringement. Mentor also asserted many more patents involving different technologies against the same emulators. A century of Supreme Court precedent holds that when a patent covers only certain features in a device, damages must be apportioned to ensure that the plaintiff is compensated only for the patent being infringed—except under very rare circumstances, such as the entire market value rule ("EMVR"), established when the patented feature drove consumer demand for the entire product. But the court permitted the jury to award damages on the entire value of the device (ultimately leading to some $36 million in damages), and did so without requiring apportionment or proof of EMVR. Because the district court failed to fix the error it acknowledged, it is incumbent on this Court to intercede.

Beyond the trial errors, this Court should also reverse the district court's drive-by invalidation of two patents asserted by Synopsys. Specifically, the court held Synopsys' '109 patent to be indefinite and its '526 patent to be patent-ineligible—and managed to dispose of each patent in two-sentence minute orders that were devoid of reasoning. In fact, the '109 patent contains ample specificity, and the '526 patent is chock-a-block with physical components that make it clearly patent-eligible.

JURISDICTION

The district court had jurisdiction under 28 U.S.C. §§ 1331, 1338, 2201. It entered final judgment on November 17, 2014. A188-92. On March 11, 2015, the court granted in part and denied in part Synopsys' post-trial motions. A193-203. Synopsys timely appealed on April 9, 2015. A36,806-08. On March 11, 2015, the court entered a permanent injunction. A204-07, 36,444-49. Synopsys timely appealed on March 16, 2015. A36,442-43, 36,457-59. This Court consolidated the appeals, Dkt. 32, and has jurisdiction. 28 U.S.C. §§ 1292(a)(1), (c)(1), 1295(a)(1).

STATEMENT OF ISSUES

1. Mentor's '376 patent claims technology for automatically identifying whether particular lines of source code were executed during an emulation of the circuit design described by the source code. To accomplish this, the invention generates "instrumentation data" that cross-references the emulation results to the corresponding source code. But Synopsys' emulators do not work that way. A designer using Synopsys' emulator must herself figure out whether a particular line of source code was executed by manually analyzing the emulation results and the source code. The '376 patent describes this additional work as

"difficult, if not impossible," and Mentor has disclaimed it. Should the judgment of infringement be reversed?

2. When a patent claims only certain features of a multifeatured device, the patentee must apportion "damages between the patented feature and the unpatented features" unless he can prove that "the entire value of the whole machine" is "attributable to the patented feature." *Garretson v. Clark*, 111 U.S. 120, 121 (1884). The accused components are two minor features of a sophisticated emulator. Did the district court err by refusing to instruct the jury on either apportionment or EMVR, instead permitting the jury to award lost profits on the full value of the emulator?

3. Under the method claimed by Synopsys' '109 patent, the results of a circuit test will be displayed on a computer screen "near" the source code to which the results correspond. The specification explains the purpose served by displaying the two items "near" each other and provides illustrations of the requisite proximity. Synopsys' expert testified that a skilled engineer would understand what "near" meant in that context. Did the district court err invalidating this patent on the ground that the term "near" is fatally indefinite?

4. The asserted claims of Synopsys' '526 patent teach adding "instrumentation circuitry" to a microchip, then altering and monitoring signals within that circuitry to improve the process of analyzing, diagnosing, and debugging circuit errors. Did the district court err in finding that these claims are about carrier wave signals—rather than physical circuitry—and thus not patent-eligible?

STATEMENT OF THE CASE

The Electronic Design Automation ("EDA") industry provides sophisticated tools that household names like Intel, Apple, Samsung, and others use to design and test computer chips. Just as BMW would not think of beginning physical production of a new sports car without rigorously testing its design, the same is true of microchips. Such "verification" has grown ever more crucial as microchips have shrunk in size and grown in complexity. At issue here is one type of verification technique known as emulation.

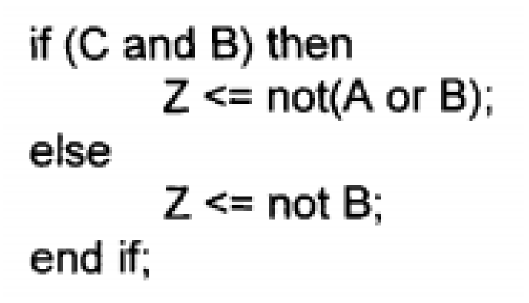
An emulator converts a chip design into a physical model of an actual chip and tests that model in various ways, including for bugs in the design. The patents here are about how to correlate circuit design test results with their associated source code—which assists the designer in fixing ("debugging") the code.

Designing Microchips And Verifying They Work Properly

We start with an overview of the chip design process, and the emulator's role in it, from coding, to synthesis, optimization, verification, and debugging.

Coding. A chip consists of innumerable tiny electronic pathways, typically implemented in silicon. The relationships among those pathways, or circuits, are what translate the behavior of electrons into the "logic" we perceive as users. Chip designers no longer draw out those circuits manually like a civil engineer planning a city. Modern integrated circuits are far too complex. Instead, designers write source code in "high-level hardware description languages" to specify how the chip will *function*—and computer programs convert those descriptions into schematics of the chip's physical components and design. A525, col. 1:15-17.

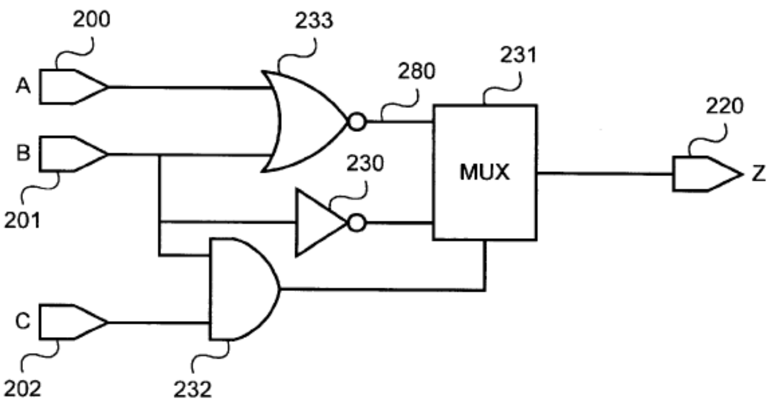
The source code used to describe the functioning of circuits is called hardware description language ("HDL"). One type of HDL is known as register transfer level ("RTL") code. A525, col. 1:15-34. Here is a simple snippet of HDL code that appears in one of Synopsys' patents:



```
if (C and B) then
        Z <= not(A or B);
else
        Z <= not B;
end if;
```

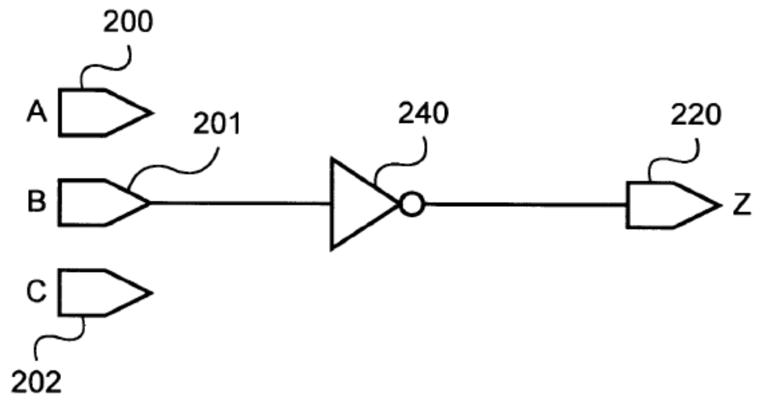
A460. This code describes how one tiny portion of the circuit should behave, using the formal logic in a common "if-then-else" statement: "[I]f" the condition is satisfied, "then" follow one branch; otherwise ("else"), follow a different branch. In this example, the value of Z may change depending on whether the condition "if (C and B)" is met. Millions of logical operations like this are built into a microchip's design, and can be mixed and matched in infinite combinations to allow a chip to perform any intended logic function. *See* A41,068.

Synthesis (or translation). Once the circuit designer writes the source code that explains how the circuit should function, the next step is to convert the functional description into a "netlist" of a chip design. To do that, the chip designer feeds the HDL code into a logic synthesis program. The program "synthesiz[es]" or "translat[es]" the code into a "netlist." A492, col. 5:4-8; A525, col. 1:35-36. A netlist can be viewed graphically as a schematic that depicts the circuit's components and the wires connecting them. Here is the schematic for the snippet of code above:



A462. The netlist's basic building blocks are "logic gates" (like 232 and 233 above). Each type of logical function described in the code is depicted by a gate with a different shape. *See generally* John F. Wakerly, *Digital Design, Principles & Practices* 76-77 (2d ed. 1994). The gates are connected by "nets," essentially microscopic wires. A526, col. 4:5-13. This version of a circuit design is known as "gate-level design."

Optimization. Often there is more than one way to achieve the designer's intended result, and the designer may not have chosen the most efficient design. Thus, after synthesis comes "optimization": The program strips unnecessary components, yielding a simpler, and often more efficient, design. A497, col. 16:39-41. Here is the optimized schematic for the same circuit:



A463. Although this optimized netlist is much simpler than the original, the two are functionally equivalent to the if-then-else statement from which they derive. A494-95, cols. 9:51-53, 12:33-42.

Optimization, however, often leads to complications for later debugging. Optimization modifies the circuit so that the source code no longer "directly correspond[s]" to the netlist. A490, col. 2:67. "Prior to optimization, it is a straight-forward task to identify which circuit element of the initial circuit corresponds to what part of the HDL source code." A492, col. 5:14-17. But "because of the extensive manipulations" during optimization, "identification after optimization becomes almost impossible." A492, col. 5:17-20. In the optimized version of the circuit, for instance, gates 232 and 233 have disappeared. Consequently, if the optimized circuit contains an error in that region, it can be "difficult, if not impossible" to pinpoint the underlying error in the source code, for lack of clear landmarks. A525, col. 2:5-9.

Verification. Verification entails a battery of tests to confirm the circuit's function, size, and speed. For "complex designs," chip designers run "millions or billions of test [patterns] … to adequately test the design." A526, col. 4:58-64. One of the techniques used to perform these tests is emulation. (Software simulation, not at issue in this appeal, is another useful technique. *See infra* 29 n.1.)

To "emulate" a design, the emulator loads the design onto an array of customizable hardware components that together imitate the behavior of a circuit. A525, col. 1:37-43. These components are called field-programmable gate arrays ("FPGAs"). FPGAs are actual working integrated circuits that are infinitely malleable, so they can be configured to incorporate a circuit design (or part of a design), and reconfigured to incorporate changes to that design—and then wiped clean to be ready to emulate the next chip. Emulating a circuit entails loading portions of the circuit design onto multiple interacting FPGAs—hundreds of FPGAs for an especially complex circuit. A41,792, 42,212. Each FPGA is typically several times larger than a typical microchip, so the emulator, with multiple FPGAs, is essentially a blown-up model of a chip.

Debugging. Emulators are good at quickly detecting that there is a bug, but pinpointing the bug's location can be a challenge for two reasons. The first is that the designer ordinarily can see only the final outputs of the circuit as a whole—i.e., whether the value of the output signal is a 0 or 1—not the internal signals, reflecting the operations of gates along the way, which exist only inside a chip. A41,100-01; A525, col. 2:20-24. Without more precision, it can be difficult to pinpoint a bug in the circuit. The second is the manner in which optimization modifies the circuit away from the source-code design. *Supra* 9-11. Thus, even when the emulator produces an erroneous test result, it can be hard to locate the bug in the corresponding source code.

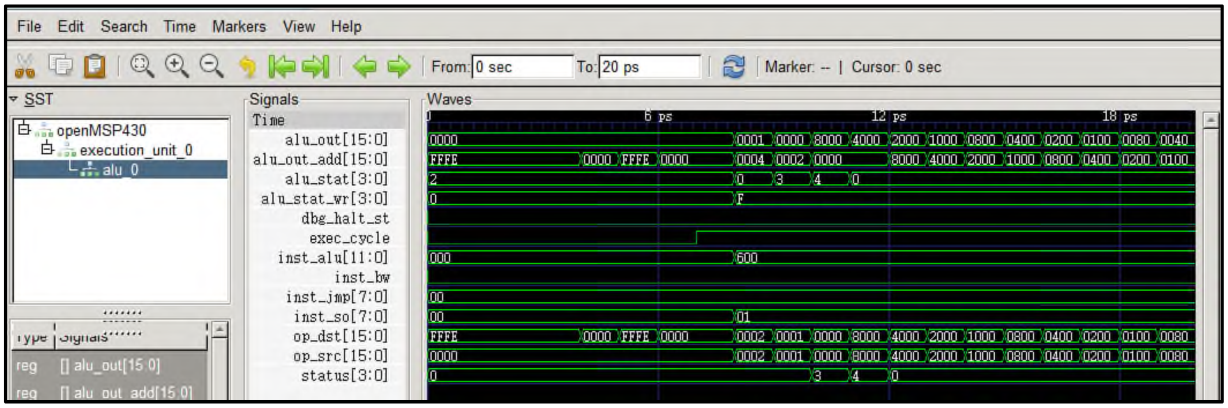
Synopsys' ZeBu Emulators, The Accused Probes

Synopsys has been an innovator in verifying and debugging chips for over 25 years. For much of its history, Synopsys provided virtually every type of tool to design and verify microchips—except for emulators. A41,376. To fill that gap, in 2012, Synopsys acquired EVE, a promising company with a pioneering emulator line. A41,432, 41,375. EVE called its emulator ZeBu for "Zero Bugs." A41,566.

While others offered expensive, high-end emulators, EVE designed a "cheaper" and "faster" alternative, with "higher capacity" to test "any" chip design, and yet smaller and less power-intensive than emulators such as Mentor's. A41,377, 41,719, 42,178; *accord* A35,831-33, 35,836 (Mentor's 30(b)(6) witness).

One of the many ways that EVE attempted to improve debug visibility was to add "flexible probes" and "value-change probes," the features Mentor accuses of infringement. These features give the designer the option to "probe" one or more internal signals—that is, to capture the values of specified signals that the designer would not normally be able to see. A45,049-51, 45,509-10, 41,947-48, 41,962, 41,972 (Synopsys' expert, Dr. Hutchings); A41,010-11, 41,049-50 (Mentor fact witness, Dr. Selvidge). To view the probed signals, the emulator collects data that is then inputted into a "waveform viewer" that translates the data into a graphical chart. A41,962-63, 45,049-51.

Here is an example of what a waveform viewer shows:

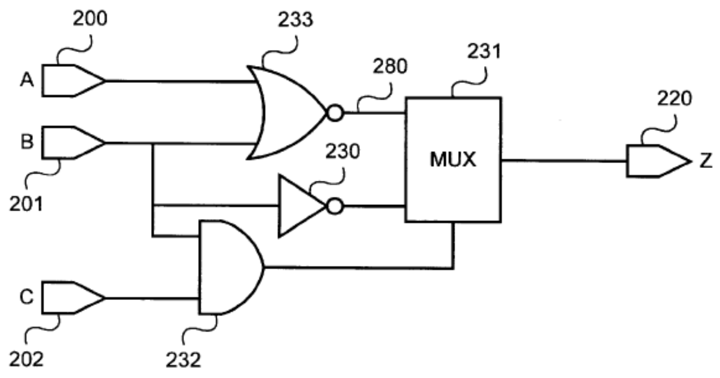


A46,383 (TX2181at\April11_2014\TESTS\April11Tests\msp\flp.vcd). In this example, the waveform viewer displays signals generated during the time period 0-20 picoseconds (ps), as indicated across the top. The "Signals" window (the middle column) lists the names assigned to the 13 signals the designer has elected to view, and the (large black) "Waves" window displays the values of those signals. This waveform viewer image shows, for example, that the signal named "exec_cycle" (the sixth one down) transitions from 0 to 1 at 7 ps.

As this example shows, a waveform viewer addresses the first complication addressed above, but not the second. It permits the designer to see internal signal values (a helpful step in finding bugs), but it does not connect the signal back to the source code to inform the designer whether a particular line of source code "executed" (i.e., ran) during the test. A41,972

(Dr. Hutchings). For a more concrete illustration, return to the exemplary "if-then" statement discussed above.

```
Lines
  1  if (C and B) then
  2           Z <= not(A or B);
  3  else
  4           Z <= not B;
  5  end if;
```



Say a test produces an incorrect value for Z. After reviewing the code and determining that signal Z appears only at lines 2 and 4 of the source code, embedded in an if-then-else statement, the designer probes signals B and C to ultimately determine whether the "then" branch (line 2) or "else" branch (line 4) was taken during the test. By probing signals B and C, the emulator can record their values and display them in the waveform viewer. The waveform viewer, however, will not tell the designer which line of code executed during the test (i.e., the "then" or the "else" branch); all the designer would know is the values of B and C. So, to know whether the test took the "then" or "else" branch, the designer would have to return to the code, analyze it using the signal values from the waveform, and then fix the bug. This particular example is almost trivially simple. If the code in question is much more complex, as it usually is, the search can be arduous. A41,883-84, 41,950-52.

Mentor And Synopsys Both Secure Patents Covering Emulation

Because it can be "difficult" to perform the additional work required to connect signal results to the corresponding source code statements, A525, col. 2:7-9, both Synopsys and Mentor set out to improve debug capability. They each patented technology directed at correlating circuit-design-test results with their associated source-code statements. Mentor asserted five patents against Synopsys, but only one survived to trial. Synopsys asserted two patents against Mentor. (Notably, the accused ZeBu emulators do not practice either of the asserted Synopsys patents.) The three patents at issue here are described in the relevant argument sections below. For present purposes, suffice it to say:

- ***Synopsys' '109 patent*** solves the problem by displaying the circuit analysis results and the corresponding source code "near" each other on a computer screen. A493, col. 7:56-59.
- ***Synopsys' '526 patent*** solves it by adding circuitry to the circuit design, then altering the signals in that modified circuit so that the results can be collected and connected with the corresponding source code. A604, 610, cols. 6:33-52, 17:35-57.
- ***Mentor's '376 patent*** solves it by generating instrumentation data that cross-references signals in the circuit design with

their associated source-code statements to indicate which source-code statements executed during the emulation test. A527, cols. 5:3-4, 5:17-45, 6:32-35.

Mentor And Synopsys Accuse Each Other Of Patent Infringement And The District Court Narrows The Case To One Patent

Six years before Synopsys acquired EVE, Mentor sued EVE, alleging that its ZeBu emulators infringed some of the same Mentor patents asserted in this case, including the '376 patent. A40,763-64, 40,774. That litigation settled when EVE took a license. But the parties agreed the license would "terminate[]" if EVE was acquired by another "company in the EDA industry." A43,040.

Shortly before Synopsys acquired EVE, Synopsys and EVE filed a declaratory judgment action, which ultimately was transferred to the District of Oregon and consolidated with two other suits that Mentor had recently brought against EVE. Synopsys later amended its complaint to allege that Mentor's "Veloce" emulators infringe Synopsys' '109 and '526 patents. For its part, Mentor asserted that EVE's (now Synopsys') ZeBu emulators infringe five of Mentor's patents, including the '376 patent. A1302-04, 1308-10, 3320-23.

Both sides moved for summary judgment on all the asserted patents except Mentor's '376 patent. The district court granted all the motions at least in part, leaving only the ’376 patent for trial. As to Synopsys’ asserted patents, the district court invalidated the ’109 patent as indefinite and the ’526 patent as directed to unpatentable subject matter. Each decision was two sentences in a minute order devoid of reasoning. A121.

The District Court Fails To Require Apportionment And Admits Its Damages Instructions Were Wrong

On damages, the district court careened from one position to another, ultimately delivering an instruction it later acknowledged was erroneous. At summary judgment, the court held that Mentor must apportion its damages to the value of the accused features (rather than obtain damages on the emulators as a whole) because the accused probes are but two of thousands of features in Synopsys’ emulators: “As to the need to apportion lost profits according to demand generated by the patented and unpatented inventions, … [a]t trial, Mentor Graphics bears the burden of showing to what extent [the ’376 patent] drove demand ….” A128.

Shortly thereafter, the court emailed the parties, repeating that Mentor “may not recover lost profits to the extent that demand … was driven by patents that have been dismissed.” A35,568. The court indicated that Mentor “may recover its lost Veloce sales in their entirety only if it can show that the ’376 patent is *the* basis for customer demand;

otherwise, demand must be apportioned among the '376 patent and those patents that have been dismissed." *Id.* In keeping with these rulings, Mentor sought leave to file a supplemental report purporting to apportion consumer demand among Mentor's five asserted patents. A24,724-27.

Then, in ruling on evidentiary motions at the final pretrial conference, the court executed an about-face: "[I]f Mentor Graphics is successful proving 'but for' causation, then … Mentor Graphics is not obligated to show apportionment" of emulator sales. A40,538. Accordingly, on the eve of trial, Mentor revised its lost-profits theory, withdrew its proposed supplemental report apportioning consumer demand, and confirmed that it would not apportion. *Id.* Mentor also expressly disavowed any intention to satisfy the EMVR exception to the general rule requiring apportionment. A40,567.

The district court took this anti-apportionment position to the extreme when it affirmatively barred Synopsys' evidence of apportionment. It ruled that Synopsys could not even challenge Mentor's claimed lost profits by demonstrating that non-accused features drove sales. A42,241 ("What's not appropriate is to say that 90 percent of the purchase price really had nothing to do with the patented feature. That's apportionment. You can't do that."); A42,242 (precluding Synopsys from contending that Mentor "shouldn't get lost profits for the whole sale price[] because the whole sale price was driven by other features"). Mentor never tried to prove EMVR. A41,523-25. Rather, the court issued jury instructions that permitted Mentor to recover lost profits for the full value of the emulators merely by showing demand for the entire *product*, without any need to apportion demand to the patented *features*. A163-64. Based on these instructions, the jury awarded Mentor over $36 million in lost profits. A184.

After trial, the court denied any "real flip or switch of positions." A42,599. Still, the district court, in response to Synopsys' post-trial briefing, acknowledged "an error in the trial" in its damages instructions. A42,600. The court conceded that it was required to instruct the jury that Mentor could not secure lost profits unless it satisfied the EMVR exception to apportionment. *Id.* (accepting that EMVR "is really a two-step process, that one first has to satisfy ['but for' causation], and then … [EMVR] as the second step"). But the court declined to order a retrial, concluding that its error was harmless because EMVR was satisfied (even after Mentor disclaimed it), merely because the patented and unpatented features were sold together. A203.

The District Court Denies Judgment As A Matter Of Law And Enters Judgment

Following trial, Synopsys renewed its motion for judgment as a matter of law. A35,380. Synopsys argued that the undisputed evidence was that the accused probes do not automatically indicate which lines of code executed during the test; instead, the designer must work backwards from the waveform viewer to deduce that information. A35,380-89. This is the argument that the district court deflected "to the Federal Circuit" without addressing the merits. A42,611-12. The court then entered an injunction against Synopsys based on the infringement judgment. A204-07.

SUMMARY OF ARGUMENT

I.A. The '376 patent automatically identifies the line(s) of source code executed during emulation, so if there is a bug, the designer will know where in the code to look for the bug. It does so by generating "instrumentation data" that cross-references the emulation test results with the corresponding source-code statements—but the key for this appeal is that the designer does not have to perform extra work to correlate the test result with the specific line(s) of code. That essential element is evident from the ordinary language of the asserted claims. Claims 1 and 28 produce a signal "*indicative* of an *execution status*" of a particular source-code statement. A533, cols. 17:61-18:7 (emphasis added). Claims 24, 26, and 27 claim a method of "*identifying* the process as *active*" during the test. A533, cols. 17:37-18:7 (emphasis added).

This Court has held that the ordinary and customary meaning of "indicating" or "identifying" certain information is to specify or point to *the very information to be identified*—not to provide *other* data that might indirectly help someone figure out the needed information. Synopsys' emulators do not "indicate" or "identify" which code executed or was active during the test. It is undisputed that Synopsys' emulators provide the designer with only the values of particular signals, and the designer must then perform additional work to figure out which line(s) of code executed during the test in order to debug the code. Because Synopsys' emulators do not "indicate" or "identify" whether particular source code has executed or is active, they do not infringe.

This common understanding of "indicate" and "identify" is confirmed by the purpose of the patent and Mentor's statements to the PTO to escape invalidity. The patent's premise is that it "can be difficult if not impossible" for the designer to work backwards to connect "signal values" from a test showing a bug back to the corresponding "source code lines." A525, col. 2:7-9. To solve this problem, the claimed invention automatically identifies the line or lines of source code that were executed during the emulation test. Thus, during *inter partes* review, Mentor argued—and the Board agreed—that Synopsys' prior art could not anticipate because the prior art's instrumentation signals "*only indicate the value of a signal* on a given net," not the code's execution status. A26,375. Having "relinquish[ed] subject matter to distinguish a prior art reference," Mentor "cannot … escape" that disavowal now. *Spectrum Int'l v. Sterilite Corp.*, 164 F.3d 1372, 1378-79 (Fed. Cir. 1998).

B. It was also improper to preclude Synopsys from contesting the '376 patent's validity based on assignor estoppel, when assignor estoppel is a dead letter under Supreme Court precedent.

II.A. The Supreme Court has long held that when a patent covers only certain features in a multicomponent device, damages must normally be apportioned to capture only the incremental value deriving from the invention. This Court has consistently followed those precedents. Among its other virtues, apportionment ensures patentees receive no more damages than "adequate to compensate for the infringement," 35 U.S.C. § 284, rather than a windfall.

B. The district court erred by failing to require Mentor to apportion. The exclusion of apportionment from the trial was particularly damaging because the accused probes are just two limited features out of thousands in the emulators and features other than the accused probes drove customer demand. In concluding that apportionment was unnecessary, the district court relied on *Rite-Hite Corp. v. Kelley Co.*, 56 F.3d 1538 (Fed. Cir. 1995) (en banc). But even if such a decision could overrule over a century of Supreme Court precedent, *Rite-Hite* did not involve a multicomponent product. Moreover, *Rite-Hite* is about "but for" causation, and thus clarifies what a plaintiff must show to qualify for *any* lost profits, while traditional apportionment principles still control the amount of lost profits—as this Court's post-*Rite-Hite* decisions demonstrate.

C. After trial, the district court acknowledged the "error in the trial" as to apportionment, A42,600, but compounded its error by upholding Mentor's unapportioned lost-profits award on the basis of EMVR—specifically, concluding that EMVR is satisfied anytime patented and unpatented components are sold together. Because the jury was never instructed on EMVR, the verdict could not be sustained on that un-instructed basis, anyway. Moreover, Mentor repeatedly and expressly disavowed reliance on EMVR, and did not attempt to (and could not) show that flexible and value-change probes were *the cause* of consumer demand for the emulators, as EMVR requires.

III. The district court further erred by invalidating Synopsys' '109 patent as indefinite in a mere two sentences. The '109 patent makes it easier to connect test results to their corresponding source code by displaying the results on screen "near" the relevant source code, so the designer knows the two are connected. The patent's illustrations depict the layout of a display, and the specification explains that the patent helps designers "trace" bugs by "displaying the results of synthesized circuit analysis" together with "the HDL source specification that generated the circuit." A493, col. 7:47-59. Based on those disclosures, a person of ordinary skill would know "with reasonable certainty … the scope of the invention." *Nautilus, Inc. v. Biosig Instruments, Inc.*, 134 S. Ct. 2120, 2124 (2014). Synopsys' expert so testified, thus at a minimum creating a triable issue of fact.

IV. The district court also erred in invalidating Synopsys' '526 patent in an eight-word opinion. The '526 patent increases debug capability by adding test circuitry to a circuit design, altering and testing the signals in that modified circuitry, and connecting the results to the corresponding source code. The patent is chock-full of complex, machine-implemented technology, presented as a standard *Beauregard*, computer-readable medium claim. The district court evidently invalidated the claim because the specification provides that, while "[p]ortions of the invention are preferably implemented in software," they can be implemented in, among other things, carrier waves, A627, col. 52:28-36, and this Court has held that "transitory electrical and electromagnetic signals" are unpatentable under 35 U.S.C. § 101, *see In re Nuijten*, 500 F.3d 1346, 1352 (Fed. Cir. 2007). But *Nuijten* is an extreme example of naked signals without more, and thus is easily distinguished here. And even if it were not easily distinguished, the district court erred in invalidating the entire claim when a single, non-preferred embodiment was potentially invalid.

STANDARD OF REVIEW

Under governing Ninth Circuit law, grants of summary judgment and denials of motions for judgment as a matter of law are reviewed de novo. *Leonel v. Am. Airlines, Inc.*, 400 F.3d 702, 708 (9th Cir. 2005); *Weaving v. City of Hillsboro*, 763 F.3d 1106, 1111 (9th Cir. 2014).

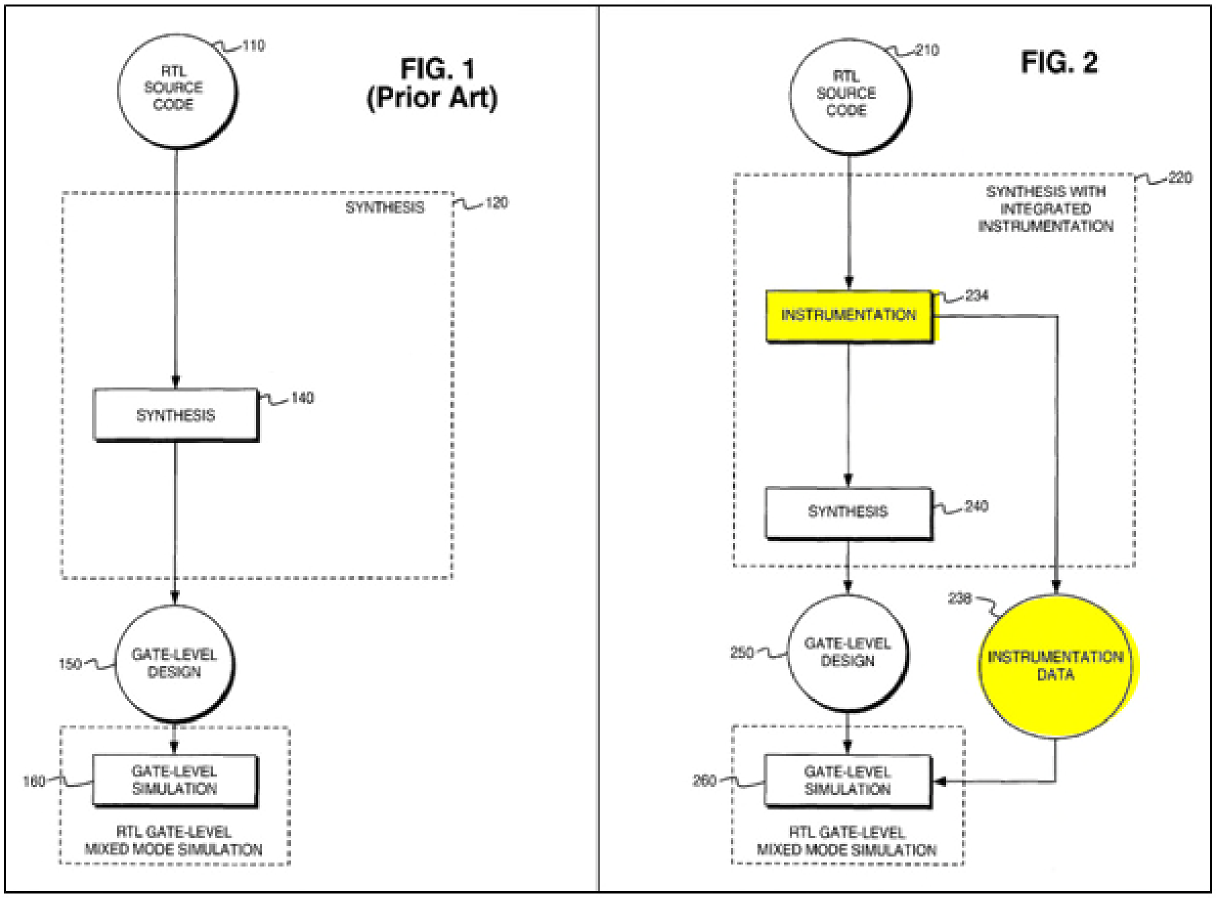
"Whether lost profits are legally compensable in a particular situation is a question of law" reviewed de novo. *Siemens Med. Solutions USA, Inc. v. Saint-Gobain Ceramics & Plastics, Inc.*, 637 F.3d 1269, 1287 (Fed. Cir. 2011). This Court also reviews de novo "the legal sufficiency of a jury instruction on an issue of patent law." *Ericsson, Inc. v. D-Link Sys., Inc.*, 773 F.3d 1201, 1225 (Fed. Cir. 2014).

ARGUMENT

I. The District Court Erred In Holding Mentor's '376 Patent Valid And Infringed.

Mentor's '376 patent teaches particular ways of improving debug visibility by automatically identifying whether specific lines of source code are executed during emulation testing. Recall that (as discussed at 9-11), optimization strips out excess circuitry and thereby loses the direct correspondence between the source code and the netlist component. The '376 patent addresses this loss of correspondence from optimization (and other causes) by generating "instrumentation data" that "cross-reference[s]" lines of source code with their "associated signal" in an emulated circuit design. A529, col. 9:15-25; *accord* A527, col. 6:32-35. This cross-reference automatically identifies the line(s) of source code executed during the emulation test. A527, col. 6:2-35.

Specifically, the patent teaches a process called “instrumentation,” which identifies source-code statements and their components in the circuit design, and “preserv[es] high-level information through the synthesis process” that otherwise would be optimized away. A527, col. 5:3-4. The patent illustrates this by contrasting the prior art with the claimed solution:



A503-04 (highlighting added); *accord* A525-28, cols. 2:30-52, 5:9-45, 7:9-38.[1] The source code is "instrumented" (234) during synthesis. That generates information about that code, called "instrumentation data" (238). This instrumentation data makes it possible for the emulator to indicate what the patent calls the "execution status" of particular lines of source code—i.e., whether particular lines of code (or "processes") were executed during the test. That way, the designer knows the universe of source code that could be responsible for particular test results during emulation. A532-33, cols. 15:2-9, 17:61-18:7.

Mentor asserted claims 1, 24, and 28 (as well as claims 26 and 27, which depend from claim 24). These claims require that the emulator "indicat[e]" or "identify[]" lines of source code responsible for a bug by specifying which lines executed during the test.

Claims 1 and 28 both revolve around whether "the instrumentation signal is *indicative* of an execution status":

[1] The '376 patent (including Figures 1 and 2) occasionally refers to "simulation." In the old days, emulation was called "hardware simulation," though "simulation" now more commonly refers to computer software that predicts the behavior of circuits. A525, col. 1:35-43. At Mentor's urging, the district court construed "simulating" to encompass both software simulation and hardware emulation. A10,847.

1. A method comprising the steps of:

   a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and

   b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is *indicative of an execution status* of the at least one statement.

28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code,

   wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal,

   wherein the instrumentation signal is *indicative of an execution status* of at least one synthesizable statement of the source code.

A532-33, cols. 15:2-9, 17:61-18:7 (emphasis added). Both claims start by synthesizing particular source code into a netlist containing an "instrumentation signal." As explained above (at 27-29), instrumentation data is generated that cross-references instrumentation signals to their corresponding line(s) of source code. This process preserves high-level information about the source code that would normally be lost, so that the source code corresponding to the emulation results of the instrumentation signals can automatically be identified. A527-28, cols. 5:9-45, 6:32-35, 8:60-64.

Thus, the instrumentation signal is "*indicative* of an execution status" of corresponding source code.

Claim 24 (and dependent claims 26 and 27) revolves around the similar concept of "identifying" particular code "as active":

> 24. A method of simulating a gate-level design comprising the steps of:
>
> a) identifying a sensitivity list of a process;
>
> b) generating logic to identify signal events for any signal in the sensitivity list; and
>
> c) *identifying the process as active during simulation* when a signal event occurs for any signal in the sensitivity list.

A533, col. 17:37-45 (emphasis added). Breaking it down: Clause (a) entails "identifying" a function in the source code (which is called "a sensitivity list of a process"). Clause (b) calls for "generating logic" (logic gates) to monitor for signal events for that particular source code. And, most relevant here, clause (c) calls for "*identifying the process* [i.e., the specific behavior described in the source code] *as active*." A533, col. 17:37-44 (emphasis added). Mentor's expert, Dr. Sarrafzadeh, explain-ed that this third limitation is analogous to "indicat[ing] an execution status" in claims 1 and 28. *See* A42,422 (knowing "which of these two statements is active, mean[s] it is going to tell me the execution status of this statement").

A. Synopsys' flexible and value-change probes do not infringe because designers still must manually connect test signals back to the source code.

This Court should reverse the infringement verdict because it depends exclusively on an infringement theory that contravenes the ordinary and customary meaning of the claim language. The ordinary meaning of "indicating" or "identifying" lines of code is to specify those very lines of code. § I.A.1. Synopsys' probes indisputably do not do that. § I.A.2. And Mentor itself disclaimed any other reading of its claims during *inter partes* review. § I.A.3.

1. The claims do not cover a device unless it "indicates" or "identifies" the code tested.

When you say something "indicates" or "identifies" certain information, the ordinary meaning is that the device specifies or points to *that very information*. You don't "indicate" information by providing *other* data that might help you indirectly figure out the needed information. A brake "indicator" light or fuel "indicator" gauge tells you explicitly that your brakes are engaged or how full your tank is; they do not provide the raw data on angles of orientation or gas pressure readings so that you can calculate your way to the answer. When you purchase an oven that is supposed to "indicate" the temperature, you expect a readout or gauge that specifies the number of degrees, not one that reports the volume and number of molecules of gas in a container, so that you can apply Avogadro's Law to calculate the temperature. "Identifying" to the Baskin-Robbins clerk your ice cream choice means pointing to or naming *that flavor*, not setting forth your ice-cream-selection principles to let the clerk figure it out.

This Court has made the point explicitly, observing that "indicative" is "a common word with a well-known meaning," and "[i]t would be inconsistent with the plain meaning of the term … to conclude that a given value is 'indicative' of another value where an entirely separate set of information … is necessary to determine that other value." *In re Bookstaff*, ___ F. App'x ___, 2015 WL 1344663, at *2 (Fed. Cir. Mar. 26, 2015); *accord Exergen Corp. v. Wal-Mart Stores, Inc.*, 575 F.3d 1312, 1321 (Fed. Cir. 2009) (plain and customary meaning of "indication" requires displayed information to be the information desired, not information that leads to the desired information); *Gemstar-TV Guide Int'l, Inc. v. ITC*, 383 F.3d 1352, 1373 (Fed. Cir. 2004) ("identify" is "defined as to link in an inseparable fashion: make correlative with something" (citation and quotation marks omitted)).

This plain and ordinary meaning is exactly the way the '376 patent uses these terms. The patent's premise is that it "can be difficult, if not impossible" for a designer to connect "signal values" indicating a bug back "to particular source code lines." A525, col. 2:7-9. This is because "[m]ost of the high-level information" contained in "the RTL source code is lost" during optimization. A525, col. 2:1-3. So while "signals can be analyzed during" emulation, that "analysis is not readily mappable to the RTL source code." A525, col. 2:7-17. As a result, according to the '376 patent, the absence of such correlation "severely limit[s]" the designer's "ability to debug the design at the gate level." A525, col. 2:20-24. Thus, the point of the '376 patent is to relieve designers of the sometimes arduous task of working backwards from the signal values generated during testing by connecting those values to the relevant source code. Instead, the claimed methods indicate to the designer the specific line(s) of source code that executed and thus produced a particular test result. That is why the relevant claims use variations of the words "indicate" or "identify."

Thus, at issue here is whether the accused ZeBu probes expressly inform the designer that certain source code has run during the emulation—that is, "indicative of an execution status" of a particular "statement of the source code" (claims 1 and 28) or that "identif[ies]" a particular "process as active" (claims 24, 26, and 27). The district court declined to consider this issue, characterizing it as "essentially boil[ing] down to … claim construction" best left "to the Federal Circuit." A42,611-12. While the court was right that it is now an issue for this Court, it was wrong in characterizing this as an issue of "claim construction." Both parties agreed that "indicative" and "identifying" are governed by their ordinary meaning.[2] Thus, no construction was needed. A10,847. *See generally Phillips v. AWH Corp.*, 415 F.3d 1303, 1312 -13 (Fed. Cir. 2005) (en banc) (patent claims "are generally given their ordinary and customary meaning" (citation omitted)). This Court can assess whether the jury's verdict is supported by the plain terms of the claim, just as it can assess whether a verdict is supported by the plain words of any other jury instruction.

[2] *See* A27,201 (Mentor argues that "'[i]dentify' is a standard English word" and "[i]t would be inappropriate … to argue that standard English words have a meaning other than the plain and ordinary meaning"); *accord* A42,418-19.

2. The accused devices do not "indicate" or "identify" that specific code statements executed or were active during a test.

The accused probes do not "indicate" or "identify" source code within this ordinary meaning. The record on this point is clear and undisputed. Mentor has never claimed that the ZeBu emulator directly tells the user which source code executed. Mentor's only argument has been that an engineer can use a waveform viewer to identify the assigned *name of the signal* (not the line of source code) and work backwards to ultimately connect the signal to its corresponding source-code statement. *See, e.g.*, A41,182-85, 41,198, 42,417, 42,423, 42,432. As Mentor said about the ZeBu in opposing JMOL:

> Dr. Sarrafzadeh explained … how you determine which line of code in the RTL will be executed based on the output of the emulator …. So he took his RTL, and he took the output from the probes, *and he correlated the output of the probe to determine[] which line of the RTL was executed*.

A42,026 (emphasis added). What Mentor's expert Dr. Sarrafzadeh explained was that a designer *cannot* determine whether a particular source-code statement was executed on the basis of the waveform viewer alone. It cannot be done without poring over the source code itself to determine which emulation results "correspond[]" to which source code.

A41,182-85. Dr. Sarrafzadeh was shown a waveform that he himself generated when testing ZeBu's flexible and value-change probes and was asked:

> Q. [W]hich piece of this output identifies an execution status as active?
>
> …
>
> Q. Dr. Sarrafzadeh, can you tell me which one is active?
>
> A. Without having the source code in front of me?
>
> Q. Yes.
>
> A. By the viewer alone, no.

A41,183-84; *accord* A41,182-83; *see* A41,972-73 (Synopsys' expert testifies that knowing the value of a signal does not indicate to the designer what is the "corresponding RTL source code statement"). Instead, Dr. Sarrafzadeh testified: "[Y]ou look at the name of the signal, on flexible probes, for example, and *you [then] associate that back* to the RTL source." A42,417 (emphasis added); *see* A42,074 (Mentor's counsel discussing Dr. Sarrafzadeh's testimony "of relating the results from these probe signals to RTL source code"). That is exactly what the '376 patent itself describes as "difficult, if not impossible." A525, col. 2:7-9.

This Court's opinion in *Exergen*, 575 F.3d at 1320-21, is on all fours. The accused product there was a thermometer, and the asserted claim required "a display for providing an *indication* of the internal temperature." *Id.* at 1320 (emphasis added). As here, the parties did not seek, and the district court did not provide, a construction of the phrase "providing an indication," so the plain and customary meaning controlled. *Id.* at 1317. The jury found infringement. But this Court overturned the verdict based on the term's ordinary meaning. The problem was that the defendant's thermometer "display[ed] a digital readout of the patient's *oral* temperature, which is different from (and typically lower than) the patient's [internal temperature]." *Id.* at 1321. This Court rejected the plaintiff's argument that "oral temperature, while not itself an internal temperature, is nevertheless an 'indication' of internal temperature because the two temperatures can be compared and correlated to one another in a clinical lookup table." *Id.* That argument "seeks to change the ordinary meaning of 'indication,'" this Court held. *Id.*

So, too, here, an emulator is not "indicative of an execution status of" a specific "statement of the source code"—nor does it "identify" a particular "process as active"—if it does not report or point to that specific "statement of the source code" or that specific "process." It is not enough to provide *other* information that enables the user to make further calculations or to wave roughly in the general direction of the offending code. In *Exergen*'s words, the code "shown on the display *must itself* be the" code in question; "it cannot be some other value requiring further (mental) computation before arriving at the" offending source code. *Id.*

3. Mentor has disclaimed any alternative meaning of "indicates" and "identifies."

This common understanding of "indicates" and "identifies" (and their derivatives) is exactly what Mentor successfully advocated in order to escape invalidation during *inter partes* review—and it cannot now argue otherwise.

The question before the PTO was whether Mentor's '376 patent was anticipated by Synopsys' '109 patent (aka "Gregory"). As described in greater detail below (at 58-60), the '109 patent discloses inserting "probe statements" into source code so that when the code is synthesized into a netlist, there are instrumentation signals "traceably related to the source HDL." A493, col. 8:21-38; *see* A496, col. 14:20-42. Mentor could not overcome the invalidity challenge without defeating the argument that the claim is satisfied merely because the "instrumentation signals *can be used* to identify when a signal event occurs during simulation." *E.g.*, A26,359 (Synopsys' petition discussing claim 24) (emphasis added).

Mentor argued emphatically that the relevant claims of the ’376 patent are not satisfied merely by providing a signal value from which a designer could work backwards to determine whether a process is active. *See* A26,375 (arguing that the ’109 patent does not read on claim 24 because “the temporary outputs ‘temp_out’ *only indicate the value of a signal* on a given net” (emphasis added)). The Board adopted Mentor’s position when it declined to review claim 24:

> [Mentor] argues that temp_in and temp_out do not qualify as “logic to identify signal events” because *they simply indicate the value of a signal* and there is no indication in Gregory that they are used to “identify the process as active” as required by the claim. We agree with [Mentor] that [Synopsys] has not shown that Gregory describes at least the required limitation “identifying the process as active during simulation when a signal event occurs for any signal in the sensitivity list.”

A26,401-02 (emphasis added) (citations omitted).

In defending the validity of claims 1 and 28, Mentor drew a similar distinction, arguing that the ’109 patent’s netlist signals do not “indicate *the execution* status of the related HDL code, or whether the related HDL code is *active*.” A26,411; *accord* A26,408. Mentor argued that “Gregory provides no indication that, after synthesis is complete, the new labels assigned to the wire were used in any fashion … in the correlating of the wire back to the HDL code.” A26,427 (discussing “‘instrumentation signal'" limitation). And again, the Board relied upon Mentor's argument, concluding in the final written decision that it was insufficient for the '109 patent to disclose technology allowing a designer to use signal values to verify that a particular line of code had been executed; rather, the claims require the instrumentation signals to convey that information explicitly. A29,801.

Statements to the PTO serve an "important function" by providing public "notice" of what the patent "claims do not cover." *Spectrum Int'l*, 164 F.3d at 1378-79 (brackets and quotation marks omitted). "[A] patentee, after relinquishing subject matter to distinguish a prior art reference asserted by the PTO during prosecution, cannot during subsequent litigation escape reliance by the defendant upon this unambiguous surrender of subject matter." *Id.* at 1379 (citation and brackets omitted). At a minimum, Mentor's concessions before the PTO are persuasive concessions of the customary meaning of the relevant terms.

B. The district court's assignor-estoppel decision is contrary to Supreme Court precedent.

The district court blocked Synopsys' validity challenges to the '376 patent on a theory of assignor estoppel. A118. It did so because one of the inventors of what became the '376 patent was Luc Burgun; he assigned the patent application to Mentor before he left to co-found EVE; and EVE was later acquired by Synopsys, which thereafter briefly employed Burgun. *See Diamond Scientific Co. v. Ambico, Inc.*, 848 F.2d 1220, 1224 (Fed. Cir.) (assignor estoppel "prevents one who has assigned the rights to a patent (or patent application)" and those "in privity with the assignor" from contesting the validity of the patent in court), *cert. dismissed*, 487 U.S. 1265 (1988).

The Supreme Court, however, demolished the doctrinal underpinnings of assignor estoppel in the decision that abolished the comparable doctrine of licensee estoppel in *Lear, Inc. v. Adkins*, 395 U.S. 653 (1969). In logic that is equally applicable to assignor estoppel, the Supreme Court reasoned that its precedents "undermined" the basis for licensee estoppel as "inconsistent with the aims of federal patent policy." *Id.* at 663-66, 673. This is especially so here, where all the inventor assigned was a patent application. *See Westinghouse Elec. & Mfg. Co. v. Formica Insulation Co.*, 266 U.S. 342, 352-53 (1924).

To the extent this panel feels bound to follow this Court's post-*Lear* decisions applying assignor estoppel, like *Diamond Scientific*, we preserve this issue for further review, or alternatively request initial hearing en banc.

II. The Damages Award Must Be Vacated Because The District Court Did Not Require Apportionment, And Mentor Did Not Satisfy The Entire Market Value Rule.

This is the rare appeal where a trial court correctly states what jury instruction is legally required, yet decides not to give it, then admits on the record that was an error, but nevertheless declines to correct the error. *Supra* 18-20.

The court was right the first time (and the third). The Supreme Court has made clear that apportionment of damages is the norm—including in lost profits cases. § II.A. This case was no exception, and the district court erred in failing to instruct the jury to apportion. § II.B. The district court only compounded the error by failing to vacate the damages award upon acknowledging its instructional error. § II.C.

A. Apportioning damages to patented features is the norm.

While the district court characterized this Court's damages jurisprudence as a "hot mess," A41,997, this case is governed by axioms that are simple and sensible. More than a century ago, the Supreme Court pronounced the rule that a patentee "*must* [satisfy] in *every* case" where "a patent is for an improvement, and not for an entirely new machine or contrivance." *Garretson*, 111 U.S. at 121 (emphasis added). That is, the patentee must apportion: "[T]he patentee … must give evidence tending to separate or apportion the defendant's profits and the patentee's damages between the patented feature and the unpatented features." *Id.* The rule is subject to the EMVR exception by which "the patentee must … show, by equally reliable and satisfactory evidence, that the profits and damages are to be calculated on the whole machine, for the reason that the entire value of the whole machine, as a marketable article, is properly and legally attributable to the patented feature." *Id.*

Principles of apportionment play an especially vital role in this age of complex, multicomponent electronic devices, *see LaserDynamics, Inc. v. Quanta Computer, Inc.*, 694 F.3d 51, 67 (Fed. Cir. 2012), though the Supreme Court recognized them more than 160 years ago. In *Seymour v. McCormick*, the Supreme Court reversed a trial court for permitting a plaintiff to recover all profits attributable to the defendant's machine, even though the patent was for an "improvement on [the] machine"—not for the "entire machine." 57 U.S. (16 How.) 480, 491 (1853). In words that could have been written for this case, the Court held it "a very grave error to instruct a jury 'that as to the measure of damages the same rule is to govern, whether the patent covers an entire machine or an improvement.'" *Id.* This failure to instruct the jury to apportion, the Court concluded, led the jury to render an "enormous and ruinous verdict" when the asserted patent was "for an improvement of small importance when compared with the whole machine." *Id.*

The Supreme Court has repeated the axiom time and again. *See Dowagiac Mfg. Co. v. Minn. Moline Plow Co.*, 235 U.S. 641, 646 (1915) ("In so far as the profits from the infringing sales were attributable to the patented improvements they belong[] to the plaintiff, and in so far as they were due to other parts or features they belong[] to the defendants."); *Westinghouse Elec. & Mfg. Co. v. Wagner Elec. & Mfg. Co.*, 225 U.S. 604, 614-15 (1912) (if a defendant's machine contains numerous components that "each may have jointly, but unequally, contributed to the profits," and "plaintiff's patent only created a part of the profits, he is only entitled to recover that part of the net gains").

This Court has repeatedly embraced the axiom as well, reiterating, as recently as last year, that "apportionment is required even for non-royalty forms of damages." *Ericsson*, 773 F.3d at 1226. "When the accused infringing products have both patented and unpatented features," the jury "must ultimately 'apportion the defendant's profits and the patentee's damages between the patented feature and the unpatented features.'" *Id.* (quoting *Garretson*, 111 U.S. at 121).

The rationale for the rule is plain: A patentee is entitled to "damages adequate to compensate for the infringement." 35 U.S.C. § 284.[3] A patentee does not ordinarily deserve damages for things she did not invent. Apportionment goes hand-in-hand with the equally venerable principle that courts cannot "enlarge a patent beyond the scope of its claim[s]." *Keystone Bridge Co. v. Phx. Iron Co.*, 95 U.S. 274, 278 (1877). To permit recovery of a machine's entire value, based on infringement of a patent for a minor improvement, would do just that. Instead, "the true measure of a patentee's general damages must be the value of what was taken." *Bandag, Inc. v. Gerrard Tire Co.*, 704 F.2d 1578, 1582 (Fed. Cir. 1983).

[3] That Congress contemplated apportionment for utility patents is further confirmed by comparison to design patents. By 1886, apportionment was required for utility and design patents alike. *See Dobson v. Dornan*, 118 U.S. 10, 17 (1886). Congress amended the design-patent statute to allow recovery of the infringer's entire profit without apportionment, but did not change the damages framework for utility patents, like this one. *See Apple Inc. v. Samsung Elecs. Co.*, 786 F.3d 983, 1001 (Fed. Cir. 2015) (quoting *Nike Inc. v. Wal-Mart Stores*, 138 F.3d 1437, 1441-42 (Fed. Cir. 1998)).

Otherwise, a defendant whose machine includes patented improvements from a "dozen" different inventors could "be compelled to pay … his whole profits to each … inventor[] of some small improvement in the [machine]." *Seymour*, 57 U.S. at 490-91. Such a result would have been merely absurd and "ruinous" when grain harvesters were at issue; in an era of semiconductors covered by hundreds or thousands of patents, this result would be downright industry-crippling.

In short, the district court was right when it said (and maintained up to the eve of trial): "Mentor Graphics bears the burden of showing to what extent [the '376 patent] drove demand …." A128.

B. The district court erroneously failed to require apportionment.

Notwithstanding the clarity and pedigree of these principles, when the district court reversed course, it committed the very "grave error" the Supreme Court warned against in *Seymour*. 57 U.S. at 491. It erred three times over in (1) relieving Mentor of any burden to attribute its lost profits to the patented features, A40,538; (2) refusing to instruct the jury on apportionment, A164-76; and (3) barring Synopsys from even putting on testimony that damages should be apportioned, A42,241.

As the district court initially recognized, the foundational rule applies here in spades. The accused probes are just two minor features of emulators that comprise thousands of hardware and software features. A35,066-67, 41,782-87, 41,895, 46,224. Debugging is just one of many functions performed by ZeBu emulators. Even within the realm of debugging, ZeBu's tools are not limited to flexible and value-change probes: Other ZeBu debug features include two additional types of probes (static and dynamic), as well as a variety of triggers, checkers, and other tools that assist with detecting and isolating design bugs. A35,066-67; *accord* A41,895, 46,224. Emulators have so many functions and features that Mentor itself has over 100 active patents covering emulation technologies. A26,567.

None of this is disputed. Synopsys' and Mentor's damages experts concluded that as many as 13 "key" features associated with emulators independently drive customer demand. A35,064 (Synopsys' expert); A26,584 (Mentor's expert). And all the available evidence shows that features other than the flexible and value-change probes drove customer demand. *See, e.g.*, A41,708-09. As the corporate representative for Intel, the foremost purchaser of emulators, testified: The "overriding factor" for purchasers is price, followed by capacity and speed; debug features are not significant factors. A32,108, 32,132. And, Mentor's corporate representative agreed Synopsys' emulators outperform Mentor's in price, size, speed, and capacity. A35,648-51, 35,655, 41,377, 42,178. Flexible and value-change probes were hardly a consideration, much less a driver of demand. A32,108, 32,118, 32,132.

Just how unmoored Mentor's damages theory is from any patented technology is most vividly illustrated by Mentor's insistence on maintaining the same lost profits claim even after it dramatically revised its view of which patented technology (if any) drives consumer demand for emulators. Early in this case, Mentor maintained that consumer demand was driven by *all five* of its asserted patents. A35,635; *see* A26,557, 26,568. When the district court rejected four of those patents on summary judgment, A121-23, 14,060-62, Mentor did not recalibrate its damages claim, still seeking more than $47 million in lost profits, A41,271.

The district court's eleventh-hour decision to deny an apportionment instruction evidently was based on Mentor's argument that this Court wiped away the apportionment requirement for lost profits cases in *Rite-Hite*, 56 F.3d at 1548. A30,252, 40,541. To read *Rite-Hite* that way would mean that this Court overruled a century of Supreme Court precedent requiring apportionment—including *Seymour*, which applied the principle in a lost profits case indistinguishable from this one. 57 U.S. at 491; *see generally* 4-5 Patent Law Perspectives § 5.2[1A][a]-[b] (2015) (collecting additional cases). *Rite-Hite* did no such thing. *Rite-Hite* did not involve a multicomponent product, and the relied-upon passage from *Rite-Hite* is not about apportionment. Instead, *Rite-Hite* addressed a question not at issue here: whether a patentee can be awarded lost profits when its own products (whose sales are claimed to have declined due to the infringement) do not practice the patented technology. 56 F.3d at 1542. The answer is that it is possible, but only when the plaintiff "show[s] … that, 'but for' the infringement, it would have made the *sales* that were made by the infringer." *Id.* at 1545 (emphasis added). Under this so-called *Panduit* causation test, a patentee establishes the requisite "but for" causation if it shows "(1) demand for the patented product; (2) absence of acceptable non-infringing substitutes; (3) manufacturing and marketing capability to exploit the demand; and (4) the amount of the profit it would have made." *Id.* at 1545 (citing *Panduit Corp. v. Stahlin Bros. Fibre Works, Inc.*, 575 F.2d 1152, 1156 (6th Cir. 1978)).

Rite-Hite thus clarifies what showing the plaintiff must make to qualify for *any* lost-profits damages when its own products do not practice the patented technology. But when it comes to quantifying those damages, traditional apportionment principles still control. *Rite-Hite* does not wipe out the age-old axiom that damages must be apportioned to cover only the patentee's inventive contribution. That *Panduit* is not a substitute for apportionment is evident from *Rite-Hite* itself. If apportionment were "subsumed" in but-for causation, courts would not have to assess EMVR after weighing the *Panduit* factors—but that is exactly what *Rite-Hite* did. 56 F.3d at 1549-51.

This Court's subsequent decisions confirm that *Rite-Hite* did not adopt some new Supreme-Court-defying exception to apportionment for lost-profits cases. When multicomponent products are at issue, this Court has continued to hold that lost profits must be apportioned to the patented component—and this requirement continues to apply even after but-for causation has been shown. In *Ferguson Beauregard/Logic Controls v. Mega Systems, LLC*, this Court reversed a district court that awarded lost profits based "on evidence of sales of a device embodying features in addition to those present in the infringed … patent." 350 F.3d 1327, 1346 (Fed.

Cir. 2003). That was error, this Court held, because the award "failed to distinguish the allocation of profits that would have been made 'but for' the infringement of the … patent with the profits that could fairly be allocated to customer demand related to [other non-infringing] features." *Id.*; *accord Calico Brand, Inc. v. Ameritex Imports, Inc.*, 527 F. App'x 987, 995-96 (Fed. Cir. 2013) (vacating lost profits award for lack of evidence of apportionment). Applying *Ferguson*, Judge Grewal recently rejected a patentee's notion that apportionment is not required under *Panduit*, and found "reversible error" in the patentee's failure to apportion. *Good Tech. Corp. v. MobileIron, Inc.*, No. 5:12-cv-05826-PSG, 2015 WL 3882608, at *5 (N.D. Cal. June 23, 2015).

Thus, this Court continues to adhere to the Supreme Court's teaching that apportionment is required "in every case," *VirnetX, Inc. v. Cisco Sys., Inc.*, 767 F.3d 1308, 1326 (Fed. Cir. 2014), "even for non-royalty forms of damages," *Ericsson*, 773 F.3d at 1226. *See also id.* at 1233 (vacating and remanding damages award where district court failed to explain to the jury that "[a] patent holder should only be compensated for the approximate incremental benefit derived from his invention"); *ResQNet.com, Inc. v. Lansa, Inc.*, 594 F.3d 860, 689 (Fed. Cir. 2010) ("At all times, the damages inquiry must concentrate on compensation for the economic harm caused by infringement of the claimed invention."); *Lucent Techs., Inc. v. Gateway, Inc.*, 580 F.3d 1301, 1337 (Fed. Cir. 2009) (vacating award based on the entire value of Microsoft Outlook because "the infringing use … [was] but a very small component of a much larger software program").

C. The district court erred in failing to vacate damages upon acknowledging its instructional mistake.

Once the district court ultimately acknowledged that its damages instructions were infected with "error," A42,600, it should have vacated the damages award. The court nonetheless tried to salvage the judgment by holding that Mentor somehow managed to satisfy EMVR, without even trying. The court's rationale was that EMVR is satisfied any time patented and unpatented components are sold together as a single device.

That was error several times over. At the outset, the district court never should have considered EVMR as a basis for sustaining the lost profits award, for Mentor expressly and repeatedly disavowed any intention to prove EMVR:

- "We've also declined to pursue lost profits under the [EMVR] in this case, so it should have no application here." A41,517.
- "We are no longer seeking to prove lost profits[] pursuant to the [EMVR]." A41,403.

- "Your Honor, on … lost profits, we do not believe the entire market value theory applies." A41,741.
- "[W]e do not believe we have to specifically show that flexible probes and value change probes drove demand for Synopsys' infringing products." A41,523.

Next, EMVR could not save the verdict because Mentor never sought a jury instruction based on EMVR, A32,410-23, and the court gave no such instruction, A144-81. The jury cannot be deemed to have awarded damages on a theory on which it wasn't even instructed, particularly where Mentor has the burden of proving this theory. *See United States v. Tarallo*, 380 F.3d 1174, 1184 (9th Cir. 2004) ("Because the jury was not instructed" on a theory of liability, "on appeal the [party defending the verdict] may not rely on this new theory."); *see z4 Techs, Inc. v. Microsoft Corp.*, 507 F.3d 1340, 1356 (Fed. Cir. 2007) ("Because no § 271(f) issues were presented to the jury, and because the jury instructions communicated the requirements for finding infringement only under § 271(a), we must assume that the jury properly confined its analysis and ultimate finding of liability to the instructions given under § 271(a)."); *Ericsson*, 773 F.3d at 1228 (vacating damages award where district court did not "separately caution the jury about the importance of apportionment").

Moreover, the district court's focus on whether patented and unpatented components are sold together bears no resemblance to the stringent showing that EMVR actually requires. EMVR is a "narrow exception" to the rule that damages must be apportioned to the patented feature. *LaserDynamics*, 694 F.3d at 67. It permits a patentee to forgo apportionment only upon showing that "the entire value of the whole machine, as a marketable article, is properly and legally attributable to the patented feature." *VirnetX*, 767 F.3d at 1326 (quoting *Garretson*, 111 U.S. at 121); *see AstraZeneca AB v. Apotex Corp.*, 782 F.3d 1324, 1337 (Fed. Cir. 2015) (patentee can obtain "damages based on the entire market value of the accused product *only* where the patented feature creates *the basis for customer demand* or *substantially creates the value of the component parts*" (emphasis added) (quoting *Uniloc USA, Inc. v. Microsoft Corp.*, 632 F.3d 1292, 1318 (Fed. Cir. 2011)); *accord LaserDynamics*, 694 F.3d at 67-68; *Good Tech. Corp.*, 2015 WL 3882608, at *4.

The district court did not conclude otherwise. It did not even consider whether the accused probes drove demand. Instead, it relied on statements in a couple of older cases that it understood to mean that EMVR is satisfied whenever the unpatented components are typically sold together with the patented components. *See* A202-03 (citing *State Indus., Inc. v. Mor-Flo Indus., Inc.*, 883 F.2d 1573, 1580 (Fed. Cir. 1989); *Kori Corp. v. Wilco Marsh Buggies & Draglines, Inc.*, 761 F.2d 649, 656 (Fed. Cir. 1985)). If that were the law, the EMVR exception would swallow the rule. After all, apportionment is an issue only when "the accused product consists of both a patented feature and unpatented features." *AstraZeneca*, 782 F.3d at 1338. Where the patent covers "the infringing product as a whole, not a single component of a multi-component product," there simply is no place for EMVR in the analysis. *Id.*

The district court's understanding is not the law. As acknowledged in one of the cases the district court cited, EMVR applies only where "the patent related feature is the basis for customer demand." *State Indus.*, 883 F.2d at 1580. This Court intended those cases to mean nothing more than that EMVR may sometimes be satisfied where the patentee "can normally anticipate the sale of the unpatented components together with the patented components." *Kori*, 761 F.2d at 656. But that is not the end of the inquiry—and this Court has never read those cases to say that it is.

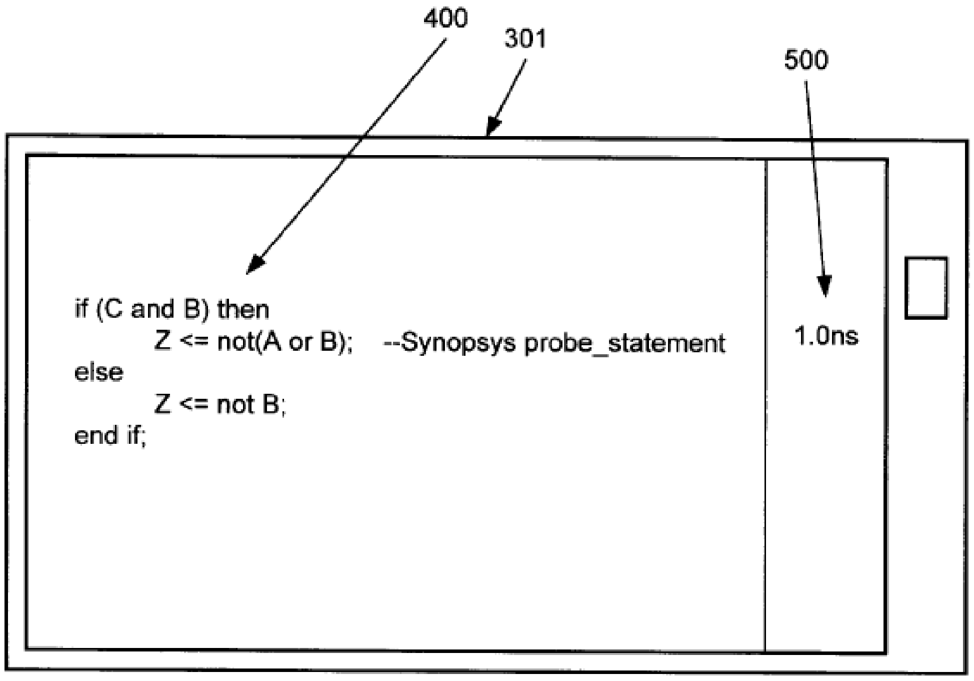
Instead, as this Court has emphasized repeatedly since deciding those cases, "[a] patentee may assess damages based on the entire market value of the accused product *only where* the patented feature creates the basis for customer demand or substantially creates the value of the component parts," and "[i]n the absence of such a showing, principles of apportionment apply." *VirnetX*, 767 F.3d at 1326; *see Good Tech. Corp.*, 2015 WL 3882608, at *5 (rejecting patentee's attempt to obtain unapportioned lost profits based merely on showing that components work together as a "single functioning unit," where patentee failed "to show that the patented features drove demand").[4] Accordingly, EMVR requires much more than selling a single product with multiple features: "[W]hen claims are drawn to an individual component of a multi-component product, it is the exception, not the rule, that damages may be based upon the value of the multi-component product." *Id.*

[4] To be sure, in a couple of cases, this Court has made statements such as: *Panduit* "does not require any allocation of consumer demand among the various *limitations* recited in a patent claim." *DePuy Spine, Inc. v. Medtronic Sofamor Danek, Inc.*, 567 F.3d 1314, 1330 (Fed. Cir. 2009); *Versata Software, Inc. v. SAP Am., Inc.*, 717 F.3d 1255, 1265 (Fed. Cir. 2013). But as the district court understood, those statements are about "*limitations* within a single patent claim" and have nothing to do with the notion that "a plaintiff may recover lost profits based on demand generated by patents not asserted in the action." A35,568 (emphasis added).

Mentor did not prove that the accused probes are the basis for customer demand, and it also erroneously convinced the court that it need not apportion damages between patented and unpatented features. The damages award therefore must be vacated.

III. The District Court Erred In Holding Synopsys' '109 Patent Indefinite.

Synopsys' '109 patent, the prior art in the *inter partes* review discussed above (at 39-41), also addresses the problem of "optimization … transform[ing] the [gate-level] design substantially," making it difficult to "trace the circuit analysis results … to the HDL source." A493, col. 7:47-54. The '109 patent solves the problem by displaying the circuit results on a computer screen "visually near" the corresponding source code. A493, col. 7:57-59. Figure 11 shows the results of a test examining how long a signal took to propagate through a small portion of the circuit:



```
if (C and B) then
        Z <= not(A or B);    --Synopsys probe_statement
else
        Z <= not B;
end if;
```

A467 (Fig. 11); *see* A494, col. 9:61-63. The circuit analysis (500) indicates that it took one nanosecond. And the designer knows which source code the test results relate to, because the results are displayed "near"—here, next to—the specific code at issue.

By displaying the results near the corresponding code, the patent associates the code with the related test results. A493, col. 7:56-59. Thus, if there is a "problem[] identified by circuit analysis," this invention "permits the designer to modify the part of the HDL specification that is responsible for" that problem. A493, col. 7:64-66. Accordingly, claim 1 covers a method that "analyz[es] said final circuit to determine characteristics associated with said final circuit's parts and … nets" and "display[s] said characteristics … *near* said portions of … [the] source text file that created said corresponding … parts and nets." A500, col. 22:34-56 (emphasis added).

Mentor sought summary judgment, arguing that claim 1 is indefinite because "[t]he word 'near' is an amorphous term of degree." A14,086-91. The district court acceded, invalidating the claim in one sentence: "The patent's claims and specification do not permit a person of ordinary skill in the art to define the claim term 'near' with reasonable certainty." A121.

"[A] patent is invalid for indefiniteness [only] if its claims, read in light of the specification delineating the patent, and the prosecution history, fail to inform, with reasonable certainty, those skilled in the art about the scope of the invention." *Nautilus*, 134 S. Ct. at 2124. Because "absolute precision is unattainable," "the definiteness requirement must take into account the inherent limitations of language" and recognize that the claim language need only provide "reasonable certainty," not absolute certainty. *Id.* at 2128-29 (quotation mark omitted).

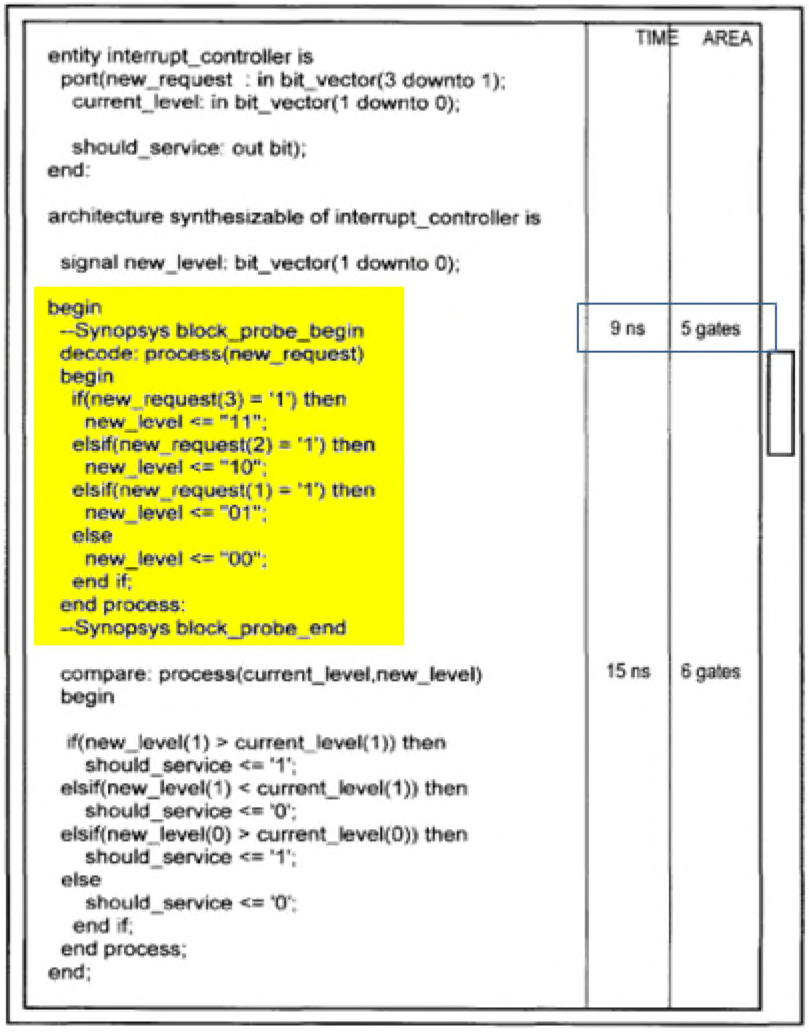
This Court has repeatedly emphasized the Supreme Court's point in holding that terms of degree are appropriate under circumstances like these. *See, e.g.*, *Biosig Instruments, Inc. v. Nautilus, Inc.* 783 F.3d 1368, 1378 (Fed. Cir. 2015) (upholding "spaced relationship") ("Claim language employing terms of degree has long been found definite where it provided enough certainty to one of skill in the art when read in the context of the invention." (quotation marks omitted)); *Deere & Co. v. Bush Hog, LLC*, 703 F.3d 1349, 1359 (Fed. Cir. 2012) ("This court has repeatedly confirmed that relative terms such as 'substantially' do not render patent claims so unclear as to prevent a person of skill in the art from ascertaining the scope of the claim."); *accord Eibel Process Co. v. Minn. & Ontario Paper Co.*, 261 U.S. 45, 65-66 (1923) (upholding claim terms "high" and "substantial elevation"); *Enzo Biochem, Inc. v. Applera Corp.*, 599 F.3d 1325, 1335 (Fed. Cir. 2010) (upholding "not interfering substantially").

In fact, this Court has approved the very term used here—"near"—reversing a district court that found it indefinite. *Young v. Lumenis. Inc.*, 492 F.3d 1336, 1345-47 (Fed. Cir. 2007). "Akin to the term 'approxi-mately,' a person having ordinary skill in the art would know where to make the cut; thus the use of the word 'near' does not deprive one of ordinary skill from being able to ascertain where the cut should be made." *Id.* at 1346. This Court rejected the notion that more precision was required when discussing the location of a medical incision on an animal "because the size of the appendage and the amount of skin required to be incised will vary." *Id.*

So, too, here. Claim 1 is clear about what must be specified, and why, and where. The goal of the invention is to make it easier for designers to "trace" the results of circuit tests and "debug[] [the] circuits efficiently." A493, col. 7:47-54; *accord* A493, 495, cols. 7:57-66, 11:29-35. It accomplishes this by "displaying the results of synthesized circuit analysis" together with "the HDL source specification that generated the circuit." A493, col. 7:57-59; *see* A490, col. 1:12-19 (displaying circuit analysis results from a netlist "in the context of the … HDL[]" to connect the two to facilitate debugging). The two types of information are displayed "near" one another, A500, col. 22:34-56 (claim 1); *accord* A455, Abstract; A493-94, cols. 7:57-59, 9:30-32, because "[c]onnecting the results of the analysis to the source" in this fashion allows the "[c]ircuit analysis results" to "be directly related to the appropriate part of the HDL source," A493, col. 8:4-10; *accord* A493 col. 8:18-20, 8:59-63.

Thus, in light of the invention's purposes and description in the specification, the patent makes clear to a skilled artisan that displaying circuit characteristics "near" the source code means doing it close enough to enable the engineer to make the mental connection between the source code and the reported test result, based on their visual placement on the computer screen. A20,530-32.

The patent's illustrations give further guidance on how "near" is near enough. *See Young*, 492 F.3d at 1346 (relying on the patent's illustrations to show the term "near" was not indefinite); *Interval Licensing LLC v. AOL, Inc.*, 766 F.3d 1364, 1373 (Fed. Cir. 2014) ("examples may satisfy the definiteness requirement"). Figure 11, reproduced above (at 59), illustrates the display of a circuit's timing-delay characteristic near the corresponding HDL code. By displaying the timing information in the column next to the related code, the patent visually associates the circuit information on screen with the corresponding code. A496, col. 13:21-22. Figure 19 provides another illustration of how "near" the two should be:



```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
    current_level: in bit_vector(1 downto 0);

    should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin
  --Synopsys block_probe_begin
  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process;
  --Synopsys block_probe_end

  compare: process(current_level,new_level)
  begin

   if(new_level(1) > current_level(1)) then
      should_service <= '1';
  elsif(new_level(1) < current_level(1)) then
      should_service <= '0';
  elsif(new_level(0) > current_level(0)) then
      should_service <= '1';
  else
      should_service <= '0';
    end if;
  end process;
end;
```

Figure 19

A475 (annotations added); *see* A494, col. 10:14-16. The test indicates that the highlighted source code produces a circuit having a physical area on the chip of five gates and a delay of nine nanoseconds. The designer knows that this information relates to the highlighted code because that test information is displayed near that source code.

In light of the invention's purpose, it would make no sense to indicate that these items had to be displayed 1 inch apart on a computer screen, or 2 inches, or 6. *See Interval Licensing*, 766 F.3d at 1370 ("absolute or mathematical precision is not required"). Just like the size and shape of the animals in *Young*, 492 F.3d at 1346, here, the size of computer screens and the amount of information displayed varies, and so it is inadvisable—if not impossible—to specify precisely how "near" the test results must be displayed to the source code.

At a bare minimum, the judgment of invalidity must be vacated and remanded because there was at least a triable issue of fact. Synopsys' expert, Dr. Hutchings, testified that "[t]he scope of near" in claim 1 "is the ability to determine what RTL code is associated with the display of information by its visual placement." A19,477; *accord* A20,530-32 (opining that an ordinarily skilled artisan would understand "near" with reasonable certainty). Mentor responded by relying heavily on the testimony of its own expert, who opined in two brief paragraphs that the '109 patent does not "define the scope or meaning of the term 'near'" or "provide any context" besides making clear that "near" "does not mean near in time," A14,228-29 (cited repeatedly by A14,086-92 (Mentor's summary-judgment brief). This disagreement between the experts precludes summary judgment. *See BJ Servs. Co. v. Halliburton Energy Servs.*, 338 F.3d 1368, 1372 (Fed. Cir. 2003) ("[D]efiniteness[] … is amenable to resolution by the jury where the issues are factual in nature."); *Teva Pharm. USA v. Sandoz Inc.*, ___ F.3d ___, 2015 WL 3772402, at *4 (Fed. Cir. June 18, 2015) ("district court's determination about how a skilled artisan would understand the way in which SEC-generated chromatogram data reflects molecular weight is a question of fact" based on expert opinion). But, far from addressing this competing testimony, and notwithstanding the strong presumption of validity for issued patents, *see generally Microsoft Corp. v. i4i Ltd. P'ship*, 131 S. Ct. 2238, 2242 (2011), the district court granted summary judgment of invalidity in a sentence without written opinion or oral explanation.

IV. The District Court Erroneously Invalidated Synopsys' '526 Patent As Patent-Ineligible.

The district court killed another of Synopsys' patents in equally summary fashion: "The patent's claims embrace unpatentable electromagnetic carrier waves." A121.

The '526 patent is yet another approach to improving debugging visibility. It teaches adding circuitry to the circuit design, then altering the signals in that modified circuit so that the results can be collected.

A604, 610, cols. 6:33-52, 17:35-57. The invention then connects those test results to the corresponding source code. A604, col. 6:33-52.

Synopsys alleged that Mentor's Veloce emulators infringe independent claims 19 and 30, and dependent claims 24, 28, and 33. A3299. Claim 19 is representative. A629, col. 55:18-40. It is a standard "*Beauregard* claim," *see In re Beauregard*, 53 F.3d 1583, 1584 (Fed. Cir. 1995), i.e., one that "formally recite[s] a tangible article of manufacture—a computer-readable medium, such as a computer disk or other data storage device—but … also require[s] the device to contain a computer program for directing a computer to carry out a specified process." *CLS Bank Int'l v. Alice Corp.*, 717 F.3d 1269, 1287 (2013) (en banc), *aff'd*, 134 S. Ct. 2347 (2014). Such claims typically are found unpatentable only where the method they disclose is drawn to an abstract idea—in *Alice Corp.*, the abstract idea was "reducing settlement risk." *Id.* at 1286. But here, Mentor made no such argument, nor could it: What claim 19 discloses is no abstract idea, but a sophisticated debugging process that operates on particular hardware.

Instead, Mentor argued, and the district court evidently agreed, that this complex, machine-implemented technology falls within the narrow and seldom-applied doctrine of *In re Nuijten*, holding that "transitory electrical and electromagnetic signals" are unpatentable. 500 F.3d at 1352; *id.* at 1353 ("transitory forms of signal transmission such as radio broadcasts, electrical signals through a wire, and light pulses through a fiber-optic cable" do not fall within the four categories of patent-eligible subject matter).

But the claims here are entirely different from the claim in *Nuijten*. *Nuijten*'s claim was drawn to "a signal," *id.* at 1351—those were the first two words of the claim—and even that extreme case posed a "difficult" question, *id.* at 1356, that divided the panel. Here, the claims are not drawn exclusively to a signal. *Compare* A629, col. 55:18-40 (claiming a "machine readable medium containing instructions" for hardware debugging). The patent is largely about hardware, with only some "portions of the invention" that are even potentially non-hardware. A627, col. 52:28-32.

Mentor's argument was that the claims are doomed by the inclusion of one item on a list, concerning an alternate embodiment, in the specification. A14,079-83. The specification says:

> Portions of the invention are *preferably* implemented in software. Such portions of the invention *can also be* embodied as computer readable code on a computer readable medium. The computer readable medium is any data storage device that can

> store data which can … thereafter be read by a computer system. Examples of the computer readable medium include read-only memory, random-access memory, CD-ROMs, magnetic tape, optical data storage devices, *carrier waves*.

A627, col. 52:28-36 (emphasis added). There is no principle of patent law that supports importing this nonexclusive example, from an alternate embodiment, to conclude that the entire claim is patent-ineligible. On the contrary, the claim language is paramount, *Phillips*, 415 F.3d at 1323 (warning against "the danger of reading limitations from the specification into the claim"), and every effort must be made to "preserve [a claim's] validity," *Tate Access Floors, Inc. v. Interface Architectural Res., Inc.*, 279 F.3d 1357, 1369 (Fed. Cir. 2002). And to the extent that one theoretical example within an alternate embodiment was patent-ineligible, and it is not, the appropriate result would be to excise that potential embodiment, not to invalidate the entire claim.

That is what makes this situation not even in the same ballpark as *Nuijten*. There, the "claims on appeal [sought] to cover the resulting encoded signals *themselves*." 500 F.3d at 1351; *accord id.* at 1356 (*Nuijten* signal was "transient," "fleeting," and "devoid of any semblance of permanence"). But, in *Nuijten*, the PTO allowed, and this Court found valid and uncontroversial, just such a claim "directed to '[a] storage medium having stored thereon a signal.'" *Id.* at 1351. That is what we have here.

In addition, this claim is far from a mere "transitory, propagating signal." *Id.* at 1357. It "contain[s] instructions" of which "portions, parts or areas of the original HDL description … are to be examined and/or modified"; it is performed on an "electronic system" with "instrumentation circuitry"; and that physical circuitry must be activated and configured. A608, 629, cols. 14:31-35, 55:18-40. For instance, the hardware to which the claims are directed includes "an electronic system having instrumentation circuitry" (claim 19) and "an integrated circuit product having instrumentation circuitry" (claim 30). The specification defines "'electronic system'" as one or more "[e]lectronic components," such as transistors and gates, that "can be implemented *entirely of hardware* … or … a mix of hardware and software." A606, col. 9:48-54 (emphasis added). The district court construed "instrumentation circuitry" to mean "'additional circuitry to facilitate debugging,'" and construed "integrated circuit product" to mean "'one or more manufactured or configured chips that implement a design in the target technology.'" A10,847.

In short, this is not the use of a generic medium that fails to add anything more to unpatentable subject matter. *See generally CLS Bank*, 717 F.3d at 1286 (Lourie, J., concurring) (cited approvingly in *Alice Corp.*, 134 S. Ct. at 2359-60). Rather, the specialized electronic system with its instrumentation circuitry is integral to the claims, and "places a meaningful limit on the[ir] scope." *SiRF Tech., Inc. v. ITC*, 601 F.3d 1319, 1332-33 (Fed. Cir. 2010); *see generally Diamond v. Diehr*, 450 U.S. 175, 188 (1981) (upholding validity of a claim incorporating an equation into a more efficient solution, even though the equation alone was not patentable). This is much more than the bare signal at issue in *Nuijten*, and the invalidation of these claims should be reversed.

CONCLUSION

This Court should reverse the district court's judgment of Synopsys' liability and damages on the '376 patent or, in the alternative, vacate and remand. This Court should also vacate its judgment of invalidity as to Synopsys' '109 and '526 patents.

Respectfully submitted,

/s/E. Joshua Rosenkranz

E. Joshua Rosenkranz
ORRICK, HERRINGTON & SUTCLIFFE LLP
51 W. 52nd Street
New York, NY 10019
(212) 506-5000

Counsel for Defendants-Appellants

July 17, 2015

ADDENDUM

TABLE OF CONTENTS

Order, Dated February 21, 2013 (DKT 472)

Nurrenbern, Noel M.

From:	info@ord.uscourts.gov
Sent:	Friday, February 21, 2014 11:18 AM
To:	nobody@ord.uscourts.gov
Subject:	Activity in Case 3:10-cv-00954-MO Mentor Graphics Corporation v. EVE-USA, Inc. et al Order on motion for partial summary judgment

U.S. District Court

District of Oregon

Notice of Electronic Filing

The following transaction was entered on 2/21/2014 at 11:17 AM PST and filed on 2/21/2014

Case Name:	Mentor Graphics Corporation v. EVE-USA, Inc. et al
Case Number:	3:10-cv-00954-MO
Filer:	
Document Number:	472(No document attached)

Docket Text:
ORDER: Mentor Graphics's motion [365] for partial summary judgment is GRANTED with respect to Mentor Graphics's assertion of assignor estoppel and DENIED in all other respects. Synopsys S.A., EVE, S.A., and EVE-USA's motion [371] for partial summary judgment will remain under advisement until after this Court issues a claim construction order. Ordered by Judge Michael W. Mosman. (dls)

3:10-cv-00954-MO Notice has been electronically mailed to:

Andrew M. Mason andrew.mason@klarquist.com, linda.schroeder@klarquist.com, litigationdocketing@klarquist.com

Aseem Saran Gupta agupta@sidley.com, rlawson@sidley.com, rpask@sidley.com, tchandler@sidley.com

David T. DeZern ddezern@sidley.com

David T. Pritikin dpritikin@sidley.com

Elizabeth Offen-Brown eoffen-brown@omm.com, drogosa@omm.com, handersen@omm.com, mkoplow@omm.com, monelson@omm.com, njanda@omm.com

Order, Dated July 25, 2014 (DKT 581)

From:	info@ord.uscourts.gov
To:	nobody@ord.uscourts.gov
Subject:	Activity in Case 3:10-cv-00954-MO Mentor Graphics Corporation v. EVE-USA, Inc. et al Order
Date:	Friday, July 25, 2014 5:01:06 PM

U.S. District Court

District of Oregon

Notice of Electronic Filing

The following transaction was entered on 7/25/2014 at 5:00 PM PDT and filed on 7/25/2014

Case Name:	Mentor Graphics Corporation v. EVE-USA, Inc. et al
Case Number:	3:10-cv-00954-MO
Filer:	
Document Number:	581(No document attached)

Docket Text:
ORDER: The Court issues a partial ruling on the parties' cross motions for summary judgment [531, 537] as follows. Mentor Graphics's Motion for Summary Judgment [531] is GRANTED as to invalidity of U.S. Patent No. 7,069,526. The patent's claims embrace unpatentable electromagnetic carrier waves. Synopsys's Motion for Summary Judgment [537] is DENIED as to assignor estoppel. Dr. Seawright's relationship to the development of Veloce's allegedly infringing features is insufficient to support a finding of privity with Mentor Graphics. Mentor Graphics's Motion for Summary Judgment [531] is GRANTED as to invalidity of U.S. Patent No. 6,132,109. The patent's claims and specification do not permit a person of ordinary skill in the art to define the claim term "near" with reasonable certainty. Synopsys's Motion for Summary Judgment [537] is GRANTED as to noninfringement of claims 2 and 4 of U.S. Patent No. 6,947,882. Mentor Graphics has not presented sufficient evidence to permit a reasonable inference that Synopsys or any of its customers has configured the FIB in a manner that allows it to route data from the design FPGAs in a module in one ZeBu unit to the design FPGAs in a module in another unit. The direct connections between ZeBu units and the design FPGAs on either end are not separate structures from the pluralities of design FPGAs

that they interconnect. The parties should prepare to present oral argument on the remaining issues in the summary judgment motions [531, 537] and Synopsys's Motion To Strike Portions of Mentor's Experts' Reply Reports [533]. Ordered by Judge Michael W. Mosman.Associated Cases: 3:10-cv-00954-MO, 3:12-cv-01500-MO, 3:13-cv-00579-MO (dls)

Case Name:	Mentor Graphics Corporation v. EVE-USA, Inc. et al
Case Number:	3:12-cv-01500-MO
Filer:	
Document Number:	353(No document attached)

Docket Text:
ORDER: The Court issues a partial ruling on the parties' cross motions for summary judgment [531, 537] as follows. Mentor Graphics's Motion for Summary Judgment [531] is GRANTED as to invalidity of U.S. Patent No. 7,069,526. The patent's claims embrace unpatentable electromagnetic carrier waves. Synopsys's Motion for Summary Judgment [537] is DENIED as to assignor estoppel. Dr. Seawright's relationship to the development of Veloce's allegedly infringing features is insufficient to support a finding of privity with Mentor Graphics. Mentor Graphics's Motion for Summary Judgment [531] is GRANTED as to invalidity of U.S. Patent No. 6,132,109. The patent's claims and specification do not permit a person of ordinary skill in the art to define the claim term "near" with reasonable certainty. Synopsys's Motion for Summary Judgment [537] is GRANTED as to noninfringement of claims 2 and 4 of U.S. Patent No. 6,947,882. Mentor Graphics has not presented sufficient evidence to permit a reasonable inference that Synopsys or any of its customers has configured the FIB in a manner that allows it to route data from the design FPGAs in a module in one ZeBu unit to the design FPGAs in a module in another unit. The direct connections between ZeBu units and the design FPGAs on either end are not separate structures from the pluralities of design FPGAs that they interconnect. The parties should prepare to present oral argument on the remaining issues in the summary judgment motions [531, 537] and Synopsys's Motion To Strike Portions of Mentor's Experts' Reply Reports [533]. Ordered by Judge Michael W. Mosman.Associated Cases: 3:10-cv-00954-MO, 3:12-cv-01500-MO, 3:13-cv-00579-MO (dls)

Case Name:	Synopsys Inc et al v. Mentor Graphics Corporation
Case Number:	3:13-cv-00579-MO
Filer:	
Document Number:	400(No document attached)

Docket Text:
ORDER: The Court issues a partial ruling on the parties' cross motions for summary judgment [531, 537] as follows. Mentor Graphics's Motion for Summary Judgment [531] is GRANTED as to invalidity of U.S. Patent No. 7,069,526. The patent's claims embrace unpatentable electromagnetic carrier waves. Synopsys's Motion for Summary Judgment [537] is DENIED as to assignor estoppel. Dr. Seawright's relationship to the development of Veloce's

allegedly infringing features is insufficient to support a finding of privity with Mentor Graphics. Mentor Graphics's Motion for Summary Judgment [531] is GRANTED as to invalidity of U.S. Patent No. 6,132,109. The patent's claims and specification do not permit a person of ordinary skill in the art to define the claim term "near" with reasonable certainty. Synopsys's Motion for Summary Judgment [537] is GRANTED as to noninfringement of claims 2 and 4 of U.S. Patent No. 6,947,882. Mentor Graphics has not presented sufficient evidence to permit a reasonable inference that Synopsys or any of its customers has configured the FIB in a manner that allows it to route data from the design FPGAs in a module in one ZeBu unit to the design FPGAs in a module in another unit. The direct connections between ZeBu units and the design FPGAs on either end are not separate structures from the pluralities of design FPGAs that they interconnect. The parties should prepare to present oral argument on the remaining issues in the summary judgment motions [531, 537] and Synopsys's Motion To Strike Portions of Mentor's Experts' Reply Reports [533]. Ordered by Judge Michael W. Mosman.Associated Cases: 3:10-cv-00954-MO, 3:12-cv-01500-MO, 3:13-cv-00579-MO (dls)

3:10-cv-00954-MO Notice has been electronically mailed to:

Andrew M. Mason andrew.mason@klarquist.com, linda.schroeder@klarquist.com, litigationdocketing@klarquist.com

Aseem Saran Gupta agupta@sidley.com, cchi@sidley.com, crowland@sidley.com, ldenbina@sidley.com, rlawson@sidley.com, rpask@sidley.com, tchandler@sidley.com

David T. DeZern ddezern@sidley.com

David T. Pritikin dpritikin@sidley.com

Elysa Q. Wan ewan@omm.com

Geoffrey H. Yost gyost@omm.com

George Riley griley@omm.com

Indra Neel Chatterjee nchatterjee@orrick.com, kmudurian@orrick.com

James C. Brooks jbrooks@orrick.com

James E. Geringer james.geringer@klarquist.com, toni.kammers@klarquist.com

Jesse Y. Cheng jcheng@orrick.com, jdeforest@orrick.com, kbarnick@orrick.com, sstillman@orrick.com

Julia E. Markley jmarkley@perkinscoie.com, asandvig@perkinscoie.com, ceffenberger-legg@perkinscoie.com, docketpor@perkinscoie.com, twoolley@perkinscoie.com

Order, Dated August 19, 2014 (DKT 589)

From:	info@ord.uscourts.gov
To:	nobody@ord.uscourts.gov
Subject:	Activity in Case 3:10-cv-00954-MO Mentor Graphics Corporation v. EVE-USA, Inc. et al Order on motion for summary judgment
Date:	Tuesday, August 19, 2014 10:03:56 AM

U.S. District Court

District of Oregon

Notice of Electronic Filing

The following transaction was entered on 8/19/2014 at 10:02 AM PDT and filed on 8/19/2014

Case Name:	Mentor Graphics Corporation v. EVE-USA, Inc. et al
Case Number:	3:10-cv-00954-MO
Filer:	
Document Number:	589(No document attached)

Docket Text:
ORDER: To the extent it remained under advisement as of my previous Order [582], Synopsys's Motion for Summary Judgment [537] is GRANTED IN PART and DENIED IN PART. As to the need to apportion lost profits according to demand generated by the patented and unpatented inventions, the motion is GRANTED. At trial, Mentor Graphics bears the burden of showing to what extent the remaining patent in issue, U.S. Patent No. 6,240,376, drove demand for Veloce emulators during the damages period. As to whether Intel is a two-supplier market, the motion is DENIED. As to capacity to meet demand, the motion is DENIED. Ordered by Judge Michael W. Mosman. (mjp)

3:10-cv-00954-MO Notice has been electronically mailed to:

Andrew M. Mason andrew.mason@klarquist.com, linda.schroeder@klarquist.com, litigationdocketing@klarquist.com

Anne E. Huffsmith ahuffsmith@omm.com

Aseem Saran Gupta agupta@sidley.com, cchi@sidley.com, crowland@sidley.com,

Email from D. L. Stephens, Dated August 25, 2014 (DKT 788-3)

From:	Dawn_Stephens@ord.uscourts.gov
Sent:	Monday, August 25, 2014 12:01 PM
To:	Mark E. Miller
Cc:	james.geringer@klarquist.com; Markley, Julia E. (Perkins Coie); #Mentor-Synopsys-Service; English, Stephen F. (Perkins Coie); 'Synopsys-Mentor@sidley.com'; Synopsys/Mentor - OHS
Subject:	RE: Mentor Graphics, Corp. EVE-USA, Inc., Civil Action No. 3:10-cv-954-MO (lead case)

Good Day Everyone,
Under the Court's Order [591] of August 19, 2014, Mentor Graphics may not recover lost profits to the extent that demand for Veloce products during the damages period was driven by patents that have been dismissed from the case. Accordingly, Mentor Graphics bears the burden of proving the extent to which the invention patented in the '376 Patent drove demand for Veloce. *DePuy Spine* and *Versata Software* are not to the contrary. They hold that the *Panduit* test does not require apportionment of demand among limitations within a single patent claim, not that a plaintiff may recover lost profits based on demand generated by patents not asserted in the action. As concerns the entire market value rule, Mentor Graphics may recover its lost Veloce sales in their entirety only if it can show that the '376 patent is *the* basis for customer demand; otherwise, demand must be apportioned among the '376 patent and those patents that have been dismissed. *See Ferguson Beauregard/Logic Controls, Div. of Dover Res., Inc. v. Mega sys., LLC*, 350 F.3d 1327, 1345–46 (Fed. Cir. 2003).

Thank you.

Dawn L. Stephens
Courtroom Deputy for Judge Michael W. Mosman
United States District Court for the District of Oregon
(503) 326-8024
Dawn_Stephens@ord.uscourts.gov

"Happiness depends more on how life strikes you than on what happens."
Andy Rooney

From: "Miller, Mark E." <markmiller@omm.com>
To: "'Dawn_Stephens@ord.uscourts.gov'" <Dawn_Stephens@ord.uscourts.gov>,
Cc: "English, Stephen F. (Perkins Coie)" <SEnglish@perkinscoie.com>, #Mentor-Synopsys-Service <MentorSynopsysService@omm.com>, "'Synopsys-Mentor@sidley.com'" <Synopsys-Mentor@sidley.com>, "'synopsys-mentor-ohs@orrick.com'" <synopsys-mentor-ohs@orrick.com>, "james.geringer@klarquist.com" <james.geringer@klarquist.com>, "Markley, Julia E. (Perkins Coie)" <JMarkley@perkinscoie.com>
Date: 08/23/2014 11:56 AM
Subject: RE: Mentor Graphics, Corp. EVE-USA, Inc., Civil Action No. 3:10-cv-954-MO (lead case)

Dear Ms. Stephens:

Mentor Graphics understands the Court's August 19 order to hold that under the "entire market value rule," Mentor Graphics must apportion its lost profits between patented and non-patented features if it is found that Mentor Graphics did not prove that the features covered by the '376 patent drove demand. Mentor Graphics disagrees with Synopsys' assertion that the order imposes an apportionment requirement into the lost profits analysis under *Panduit Corp. v. Stahlin Bros. Fibre Works, Inc.*, 575 F.2d 1152 (6th Cir. 1978). Such a requirement would be contrary to the holdings in *DePuy Spine v. Medtronic Sofamor Danek, Inc.*, 567 F.3d 1314, 1330 (Fed. Cir. 2009), and *Versata Software, Inc. v. SAP America, Inc.*, 717 F.3d 1255, 1265 (Fed. Cir. 2013).

Mentor Graphics is prepared to prove lost profits under either (1) the entire market value rule or (2) *Panduit*, neither of which, if established, requires apportionment. Mentor Graphics understands that Synopsys' motion for summary judgment on lost profits addressed only the entire market value approach (*see* Docket No. 537 at 3-4), which is an approach that obviates the need to apportion between patented and non-patented features if the patent holder ~~can~~ shows that "the patent-related feature is the 'basis for customer demand.'" *Rite-Hite Corporation v. Kelley Company, Inc.*, 56 F.3d 1538, 1549 (Fed. Cir. 1995). This is in accord with the Court's holding that Mentor Graphics bears the burden of showing "to what extent the remaining patent at issue … drove demand for Veloce emulators …"

Synopsys' motion for summary judgment did not squarely address the Panduit approach, mentioning Panduit only in passing. Well-settled Federal Circuit precedent establishes that Panduit does not require apportionment. See, *DePuy*, 567 F.3d at 1330, and *Versata*, 717 F.3d at 1265. Panduit simply requires that the patentee show that there is "demand for the patented product." *Rite-Hite*, 56 F.3d at 1545. Alternatively, the patentee may show demand for a product that "directly competes with the infringing device," *DePuy*, 567 F.3d at 1330, or demand for the infringing device itself. *Versata*, 717 F.3d at 1265-1266. The importance of the patented features is addressed by the second Panduit factor: the absence of any acceptable, non-infringing substitutes. This is also in accord with the Court's finding that Mentor Graphics bears the burden of showing the importance of the patented features, since if the patented features are not important to customers, a non-infringing alternative without such features would be acceptable. *See*, *Rite-Hite*, 567 F.3d at 1330-1331.

Mentor Graphics would appreciate clarification from the Court that (1) Mentor Graphics bears the burden of showing the importance the features covered by the '376 patent for purposes of the entire market value rule or under the second Panduit factor, and (2) apportionment is required only if Mentor Graphics fails to meet that burden.

Respectfully submitted,

James Geringer and Mark E. Miller for Mentor Graphics

Mark E. Miller
O'Melveny & Myers LLP
P: 415.984.8904
E: markmiller@omm.com

From: English, Stephen F. (Perkins Coie) [mailto:SEnglish@perkinscoie.com]
Sent: Friday, August 22, 2014 11:40 AM
To: 'Dawn_Stephens@ord.uscourts.gov'
Cc: #Mentor-Synopsys-Service; 'Synopsys-Mentor@sidley.com'; 'synopsys-mentor-ohs@orrick.com'; james.geringer@klarquist.com; Miller, Mark E.; Markley, Julia E. (Perkins Coie); English, Stephen F. (Perkins Coie)
Subject: Mentor Graphics, Corp. EVE-USA, Inc., Civil Action No. 3:10-cv-954-MO (lead case)

Dear Ms. Stephens,

Synopsys seeks guidance on the effect of the Court's ruling with respect to apportionment of lost profits addressed in its August 19, 2014 Order (Dkt. 589). The Court's August 19 order stated, "As to the need to apportion lost profits according to demand generated by the patented and unpatented inventions, the motion is GRANTED." This order forecloses Mentor's effort to present a lost profits theory that does not apportion according to demand generated by the patented and unpatented inventions. Notwithstanding this order, Mentor informed Synopsys that it still intends to seek lost profits without apportioning according to demand generated by the patented and unpatented inventions.

Mentor's untenable position will cause substantial unnecessary trial preparations, additional motions practice, and expense. Synopsys seeks to avoid that needless expense and distraction. Accordingly, Synopsys requests that the Court restate that Mentor may not present a lost profits case that does not apportion.

If the Court would find a teleconference or further written submission helpful, Synopsys would be happy to arrange a call or provide a further written submission.

Thank you in advance for your assistance on this matter.

Yours very truly,
Steve English and Neel Chatterjee for Synopsys/EVE

Stephen F. English | Perkins Coie LLP
Partner - Litigation
PHONE: 503.727.2003
FAX: 503.346.2003
CELL: 503.799.3307
E-MAIL: senglish@perkinscoie.com

Civil Minutes, Dated September 23, 2014 (DKT 674)

UNITED STATES DISTRICT COURT
DISTRICT OF OREGON

CIVIL MINUTES

Case No.: 3:10-cv-00954-MO (LEAD) **Date of Proceeding:** September 23, 2014

Case Title: Mentor Graphics Corporation v. v. EVE-USA, Inc., et al.

Presiding Judge: **Hon. Michael W Mosman** **Courtroom Deputy:** Dawn Stephens
Dawn_Stephens@ord.uscourts.gov
Telephone number (503) 326-8024

Reporter: Bonita Shumway **Tape No:** ____________

DOCKET ENTRY: RECORD OF PRETRIAL CONFERENCE:

MOTIONS IN LIMINE

ORDER GRANTING in part and DENYING in part and taken UNDER ADVISEMENT in part Plaintiff Mentor Graphics Corporation's Motions in Limine [627]. No. 1 is DENIED as stated on the record; No. 2 is GRANTED in part and DENIED in part: evidence of the '109 patent can be used to rebut willfulness but not to attack the validity of the '376 patent, there is no decision yet on whether or not Synopsys can bring up invalid Mentor Graphics patents; No. 3 is GRANTED in part and DENIED in part: Luc Burgun can testify as to Defendant's state of mind, but not on the ultimate issue of infringement; No. 4 is DENIED as stated on the record; No. 5 is GRANTED as stated on the record; No. 6 is GRANTED in part and DENIED in part: lay witnesses cannot offer opinions on the ultimate issue of actual infringement, but can testify as to Defendant's state of mind; No. 7 is DENIED as stated on the record; No. 8 is UNDER ADVISEMENT to be resolved after further briefing from the parties; No. 9 is DENIED as stated on the record; No. 10 is DENIED as stated on the record; No. 11 is DENIED as stated on the record.

ORDER GRANTING in part and DENYING in part and taken UNDER ADVISEMENT in part Defendant Synopsys Emulation and Verification S.A.'s Motions in Limine [619]. No. 1 is UNDER ADVISEMENT to be resolved after further briefing from the parties; No. 2 is DENIED as stated on the record; No. 3 is UNDER ADVISEMENT to be resolved after further briefing from the parties; No. 4 is GRANTED as stated on the record; No. 5 is DENIED as stated on the record; No. 6 is DENIED as stated on the record; No. 7 is GRANTED as stated on the record; No. 8 is GRANTED as stated on the record; No. 9 is DENIED as stated on the record; No. 10 is GRANTED as stated on the record; No. 11 is DENIED as stated on the record.

WITNESS STATEMENTS

Defendant's objection to: Plaintiff's expert witness Dr. Majid Saarafzdeh is DENIED; Plaintiff's expert witness Stephen Degnan is UNDER ADVISEMENT; Plaintiff's expert witness Ms. Suzanne Stuckwisch is UNDER ADVISEMENT.

Any additional objections to witness statements remaining in light of the Court's rulings on motions in limine and *Daubert* motions to be raised and resolved either at a hearing set on Friday September 26, 2014, or at trial.

EXHIBITS

Plaintiff's Motion to Exclude Exhibits: 2185 and 2186 is GRANTED; 2075 and 2076 is DENIED; 2072–2074, and 2170 is DENIED; 2070, 2102–13, 2139–69, 2171–75, 2182–84 is DENIED.

Defendant's Motion to Exclude Exhibits: 489–608 is DENIED; 143, 155, 170, 171, 200, 358, 360, 402–04, 407, 408, 412–14, and 674 is DENIED; 41, 75–77, 85, 130–32, 168, and 278 is DENIED to the extent that these exhibits are used only to prove that certain communications took place and not what the contents of those communications were; 308–39 is DENIED; 415–49, and 450 is DENIED to the extent they have not been withdrawn by Plaintiff.

Objections to exhibits remaining in light of the Court's rulings on motions in limine to be raised and resolved at trial.

ADDITIONAL MATTERS

Jury selection and trial management discussed.

The parties are instructed to be here at 8:45 a.m. in Courtroom 16 on the Sixteenth Floor. The trial will commence Monday, September 29, 2014, at 9:00 a.m.

PLAINTIFF'S COUNSEL	**DEFENDANTS' COUNSEL**
Mark E. Miller	Neal Chatterjee
Geoffrey H. Yost	Steve English
George Riley	Julia Markley
Anne E. Huffsmith	Scott Lonardo
James E. Geringer	Travis Jensen

cc: { } All counsel

DOCUMENT NO: ______

Minutes of Proceedings, Dated September 26, 2014 (DKT 686)

From:	info@ord.uscourts.gov
Sent:	Friday, September 26, 2014 5:21 PM
To:	nobody@ord.uscourts.gov
Subject:	Activity in Case 3:10-cv-00954-MO Mentor Graphics Corporation v. EVE-USA, Inc. et al Order on Motion - Miscellaneous

U.S. District Court

District of Oregon

Notice of Electronic Filing

The following transaction was entered on 9/26/2014 at 5:21 PM PDT and filed on 9/26/2014

Case Name: Mentor Graphics Corporation v. EVE-USA, Inc. et al
Case Number: 3:12-cv-01500-MO
Filer:
Document Number: 456(No document attached)

Docket Text:
MINUTES of Proceedings: Final Pretrial Conference held. Order GRANTING IN PART Motion in Limine #8 [627]. ORDER Donald Cantow was sufficiently disclosed as a damages witness with personal knowledge of Mentor Graphics's manufacturing capacity. Therefore Donald Cantow may testify at trial with respect to Mentor Graphics's manufacturing capacity. The parties are ordered to appear in the courtroom at 8:30am September 29, 2014 to determine the admissibility of Ms. Stuckwisch's damages testimony. The sole issue discussed will be whether or not Ms. Stuckwisch was properly disclosed as a potential damages witness for trial. Order GRANTING IN PART DENYING IN PART Motion in Limine [619]. Order DENYING #1; GRANTING #3. Motion to Exclude the Testimony of Stephen Degnan based on VirnetX, Inc. v. Cisco Systmes. Inc. [668] is DENIED. Mark Miller, Jeff Yost, George Riley, Anne E. Huffsmith and James E. Geringer present as counsel for plaintiff(s). Neal Chatterjee, Julia Markley, Steve English, Travis Jensen and Scott Lonardo present as counsel for defendant(s).(Court Reporter Dennis Apodaca.) Associated Cases: 3:10-cv-00954-MO, 3:12-cv-01500-MO, 3:13-cv-00579-MO(dls)

Case Name: Synopsys Inc et al v. Mentor Graphics Corporation
Case Number: 3:13-cv-00579-MO
Filer:

Document Number: 503(No document attached)

Docket Text:
MINUTES of Proceedings: Final Pretrial Conference held. Order GRANTING IN PART Motion in Limine #8 [627]. ORDER Donald Cantow was sufficiently disclosed as a damages witness with personal knowledge of Mentor Graphics's manufacturing capacity. Therefore Donald Cantow may testify at trial with respect to Mentor Graphics's manufacturing capacity. The parties are ordered to appear in the courtroom at 8:30am September 29, 2014 to determine the admissibility of Ms. Stuckwisch's damages testimony. The sole issue discussed will be whether or not Ms. Stuckwisch was properly disclosed as a potential damages witness for trial. Order GRANTING IN PART DENYING IN PART Motion in Limine [619]. Order DENYING #1; GRANTING #3. Motion to Exclude the Testimony of Stephen Degnan based on VirnetX, Inc. v. Cisco Systmes. Inc. [668] is DENIED. Mark Miller, Jeff Yost, George Riley, Anne E. Huffsmith and James E. Geringer present as counsel for plaintiff(s). Neal Chatterjee, Julia Markley, Steve English, Travis Jensen and Scott Lonardo present as counsel for defendant(s).(Court Reporter Dennis Apodaca.) Associated Cases: 3:10-cv-00954-MO, 3:12-cv-01500-MO, 3:13-cv-00579-MO(dls)

Case Name:	Mentor Graphics Corporation v. EVE-USA, Inc. et al
Case Number:	3:10-cv-00954-MO
Filer:	
Document Number:	686(No document attached)

Docket Text:
MINUTES of Proceedings: Final Pretrial Conference held. Order GRANTING IN PART Motion in Limine #8 [627]. ORDER Donald Cantow was sufficiently disclosed as a damages witness with personal knowledge of Mentor Graphics's manufacturing capacity. Therefore Donald Cantow may testify at trial with respect to Mentor Graphics's manufacturing capacity. The parties are ordered to appear in the courtroom at 8:30am September 29, 2014 to determine the admissibility of Ms. Stuckwisch's damages testimony. The sole issue discussed will be whether or not Ms. Stuckwisch was properly disclosed as a potential damages witness for trial. Order GRANTING IN PART DENYING IN PART Motion in Limine [619]. Order DENYING #1; GRANTING #3. Motion to Exclude the Testimony of Stephen Degnan based on VirnetX, Inc. v. Cisco Systmes. Inc. [668] is DENIED. Mark Miller, Jeff Yost, George Riley, Anne E. Huffsmith and James E. Geringer present as counsel for plaintiff(s). Neal Chatterjee, Julia Markley, Steve English, Travis Jensen and Scott Lonardo present as counsel for defendant(s).(Court Reporter Dennis Apodaca.) Associated Cases: 3:10-cv-00954-MO, 3:12-cv-01500-MO, 3:13-cv-00579-MO(dls)

3:12-cv-01500-MO Notice has been electronically mailed to:

Scott D. Eads   seads@perkinscoie.com, MTompkins@perkinscoie.com, ceffenberger-legg@perkinscoie.com, docketpor@perkinscoie.com, khuang@perkinscoie.com

Stephen F. English   senglish@perkinscoie.com, ajaclark@perkinscoie.com

Dennis P. Rawlinson (Terminated)   dennis.rawlinson@millernash.com, jane.rausch@millernash.com

James E. Geringer   james.geringer@klarquist.com, amy.uhl@klarquist.com, toni.kammers@klarquist.com

Final Jury Instructions, Dated October 9, 2014 (DKT 720)

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF OREGON

PORTLAND DIVISION

MENTOR GRAPHICS CORPORATION,
an Oregon corporation,

Plaintiff and Counter-Defendant,

v.

EVE-USA, INC., a Delaware corporation, and **SYNOPSYS EMULATION AND VERIFICATION S.A.,** formed under the laws of France,

Defendants and Counter-Claimants.

No. 3:10-cv-00954-MO (Lead)
Case No. 3:12-cv-01500-MO
Case No. 3:13-cv-00579-MO

FINAL
JURY INSTRUCTIONS

FINAL JURY INSTRUCTIONS

DATED: This 9th day of October, 2014.

/s/Michael W. Mosman
MICHAEL W. MOSMAN
United States District Judge

Instruction No. 1

DUTY OF THE JURY

Members of the Jury: Now that you have heard all of the evidence and the arguments of the attorneys, it is my duty to instruct you as to the law of the case.

A copy of these instructions will be sent with you to the jury room when you deliberate.

You must not infer from these instructions or from anything I may say or do that I have an opinion regarding the evidence or what your verdict should be.

It is your duty to find the facts from all the evidence in the case. To those facts you will apply the law as I give it to you. You must follow the law as I give it to you whether you agree with it or not. And you must not be influenced by any personal likes or dislikes, opinions, prejudices, or sympathy. That means that you must decide the case solely on the evidence before you. You will recall that you took an oath to do so.

In following my instructions, you must follow all of them and not single out some and ignore others. They are all important.

Instruction No. 2

PREPONDERANCE OF THE EVIDENCE

When a party has the burden of proof on any claim or affirmative defense by a preponderance of the evidence, it means you must be persuaded by the evidence that the claim or affirmative defense is more probably true than not true.

You should base your decision on all of the evidence, regardless of which party presented it.

Instruction No. 3

WHAT IS EVIDENCE

The evidence you are to consider in deciding what the facts are consists of:

(1) the sworn testimony of any witness;

(2) the exhibits which are received into evidence; and

(3) any facts to which the lawyers have agreed.

Instruction No. 4

WHAT IS NOT EVIDENCE

In reaching your verdict, you may consider only the testimony and exhibits received into evidence. Certain things are not evidence, and you may not consider them in deciding what the facts are. I will list them for you:

(1) Arguments and statements by lawyers are not evidence. The lawyers are not witnesses. What they have said in their opening statements, closing arguments, and at other times is intended to help you interpret the evidence, but it is not evidence. If the facts as you remember them differ from the way the lawyers have stated them, your memory of them controls.

(2) Questions and objections by lawyers are not evidence. Attorneys have a duty to their clients to object when they believe a question is improper under the rules of evidence. You should not be influenced by the objection or by the court’s ruling on it.

(3) Testimony that has been excluded or stricken, or that you have been instructed to disregard, is not evidence and must not be considered. In addition, sometimes testimony and exhibits are received only for a limited purpose. When I have given a limiting instruction, you must follow it.

(4) Anything you may have seen or heard when the court was not in session is not evidence. You are to decide the case solely on the evidence received at the trial.

Instruction No. 5

Direct and Circumstantial Evidence

Evidence may be direct or circumstantial. Direct evidence is direct proof of a fact, such as testimony by a witness about what that witness personally saw or heard or did. Circumstantial evidence is proof of one or more facts from which you could find another fact. You should consider both kinds of evidence. The law makes no distinction between the weight to be given to either direct or circumstantial evidence. It is for you to decide how much weight to give to any evidence.

Instruction No. 6

CREDIBILITY OF WITNESSES

In deciding the facts in this case, you may have to decide which testimony to believe and which testimony not to believe. You may believe everything a witness says, or part of it, or none of it. Proof of a fact does not necessarily depend on the number of witnesses who testify about it.

In considering the testimony of any witness, you may take into account:

1) the opportunity and ability of the witness to see or hear or know the things testified to;
2) the witness's memory;
3) the witness's manner while testifying;
4) the witness's interest in the outcome of the case and any bias or prejudice;
5) whether other evidence contradicted the witness's testimony;
6) the reasonableness of the witness's testimony in light of all the evidence; and
7) any other factors that bear on believability.

Instruction No. 7

EXPERT OPINION

You have heard testimony from persons who, because of education or experience, are permitted to state opinions and the reasons for their opinions.

Opinion testimony should be judged just like any other testimony. You may accept it or reject it, and give it as much weight as you think it deserves, considering the witness's education and experience, the reasons given for the opinion, and all the other evidence in the case.

Instruction No. 8

DEPOSITION TESTIMONY OF A 30(b)(6) DESIGNEE

Certain witnesses presented during this trial have been referred to by counsel as “30(b)(6) witnesses” or “30(b)(6) designees.” In litigation, a party may request the deposition of a corporation or other organization by serving the corporation or organization with a deposition notice or subpoena, describing with reasonable particularity the matters on which the corporation or organization will be examined. The corporation or organization must then designate one or more officers, directors, managing agents, or other persons to testify on its behalf. These persons are referred to as “30(b)(6) witnesses” or “30(b)(6) designees.” A 30(b)(6) witness must testify during his deposition about information known or reasonably available to the corporation or organization regarding the matters on which he was designated to testify.

The testimony of a 30(b)(6) witness regarding the matters on which he was designated to testify constitutes the testimony of the corporation or organization. You should give this testimony the same weight that you would give other testimony presented at trial. This testimony may be contradicted by other testimony or evidence presented at trial and you must decide which testimony or evidence is accurate.

Instruction No. 9

PRIVILEGED INFORMATION

A party does not need to disclose privileged information. Privileged information includes, for example, (1) confidential communications between a client and her attorney, and (2) work product created by an attorney in anticipation of litigation.

A party may not, however, rely on privileged information to show a good-faith belief in non-infringement or invalidity of a patent. If a party asserts that it relied on the advice or investigation of its attorneys (including attorneys within a corporation) to form its belief that a patent is invalid or not infringed, that party waives its privilege and must disclose its attorney's opinion, advice, investigation, and all other information relating to the subject of the investigation. In this case, Synopsys has claimed privilege and has not disclosed any attorney opinion, advice, or investigation of non-infringement or invalidity of the '376 Patent. Synopsys cannot, therefore, rely on such privileged information to support any non-infringement argument in this trial.

Instruction No. 10

SUMMARY OF CONTENTIONS

As I did at the start of the case, I will first give you a summary of each side's contentions in this case. I will then provide you with detailed instructions on what each side must prove to win on each of its contentions.

As I previously told you, Mentor Graphics contends that Synopsys infringes the '376 patent. The asserted claims of the '376 patent are: 1, 24, 26, 27, and 28. Mentor Graphics also argues that Synopsys has actively induced infringement of these claims of the '376 patent by others, and contributed to the infringement of these claims of the '376 patent by others. The emulation systems at issue are ZeBu-Server, ZeBu-Server 2, ZeBu-Server 3, and ZeBu-Blade. These products may be referred to collectively as the "accused products," the "accused emulation systems," or "Synopsys' emulation systems."

The validity and enforceability of the '376 patent is not at issue in this trial. For purposes of this trial, the '376 patent is valid and enforceable.

Synopsys denies that it has infringed the asserted claims of the '376 patent and denies that it has induced or contributed to any infringement of the '376 patent. Synopsys contends that its accused ZeBu emulation systems do not infringe the '376 patent because neither Synopsys nor its customers perform every step of claims 1, 24, 26, and 27 of the '376 patent and its accused ZeBu emulation systems do not meet all the elements of claim 28 of the '376 patent.

Your job is to decide whether Synopsys has infringed the asserted claims of the '376 patent. If you decide that any claim of the '376 patent has been infringed, you will then need to decide any money damages to be awarded to Mentor Graphics to compensate it for the infringement.

Instruction No. 11

THE PATENT AT ISSUE

I have already determined the meaning of certain claim terms of the ’376 patent. You will be been given a document reflecting those meanings. For a claim term for which I have not provided you with a definition, you should apply the ordinary meaning. You are to apply my definitions of these terms throughout this case. However, my interpretation of the language of the claims should not be taken as an indication that I have a view regarding issues such as infringement of the ’376 patent. Those issues are yours to decide.

Instruction No. 12

The Role of the Claims of a Patent

Before you can decide many of the issues in this case, you will need to understand the role of patent "claims." The patent claims are the numbered sentences at the end of each patent. The claims are important because it is the words of the claims that define what a patent covers. The figures and text in the rest of the patent provide a description and/or example of the invention and provide a context for claims, but it is the claims that define the breadth of the patent's coverage. Each claim is effectively treated as if it were a separate patent, and each claim may cover more or less than another claim. Therefore, what a patent covers depends, in turn, on what each of its claims covers.

You will first need to understand what each claim covers in order to decide whether or not there is infringement of the claim. The law says that it is my role to define the terms of the claims. You will be given a document that includes my definitions. You must accept my definitions of these words in the claims as being correct. It is your job to take these definitions and apply them as you consider whether Synopsys infringes the asserted patent claims.

I instruct you that the following claim terms have the following definition, all of which are also included in the document you will be given:

Claim Term	Construction
gate-level netlist	"a list of gates and their inputs, outputs, and interconnects"
gate-level design	"a list of gates and their inputs, outputs, and interconnects"
instrumentation signal	"output signal added or preserved during logic synthesis that indicates whether a corresponding RTL source code statement is active"
simulating a gate-level design	"modeling the operation of a gate level design using software and/or hardware"
sensitivity list	"a list of signals to which a process is responsive"
process	"a description of the behavior of some portion of a circuit design"
generating logic	"generating gates"

If I have not provided a specific definition for a given term, you are to use the ordinary meaning of that term.

Instruction No. 13

HOW A CLAIM DEFINES WHAT IT COVERS

I will now explain how a claim defines what it covers.

A claim sets forth, in words, a set of requirements. Each claim sets forth its requirements in a single sentence. If a device or a method satisfies each of these requirements, then it is covered by the claim.

There can be several claims in a patent. Each claim may be narrower or broader than another claim by setting forth more or fewer requirements. The coverage of a patent is assessed claim-by-claim. In patent law, the requirements of a claim are often referred to as "claim elements" or "claim limitations." When a product meets all of the requirements of a claim, the claim is said to "cover" that thing, and that thing is said to "fall" within the scope of that claim. In other words, a claim covers a product or process where each of the claim elements or limitations is present in that product or process.

Sometimes the words in a patent claim are difficult to understand, and therefore it is difficult to understand what requirements these words impose. It is my job to explain to you the meaning of the words in the claims and the requirements these words impose. As I just instructed you, there are certain specific terms that I have defined and you are to apply the definitions that I provide to you.

By understanding the meaning of the words in a claim and by understanding that the words in a claim set forth the requirements that a product or process must meet in order to be covered by that claim, you will be able to understand the scope of coverage for each claim. Once you understand what each claim covers, then you are prepared to decide whether Mentor has proven that Synopsys infringes the asserted claims of the '376 patent.

Instruction No. 14

INDEPENDENT AND DEPENDENT CLAIMS

This case involves two types of patent claims: independent claims and dependent claims.

An “independent claim” sets forth all of the requirements that must be met in order to be covered by that claim. Thus, it is not necessary to look at any other claim to determine what an independent claim covers. In this case, claims 1, 24, and 28 of the ’376 patent are each independent claims.

Claims 26 and 27 of the ’376 patent are “dependent claims.” A dependent claim does not itself recite all of the requirements of the claim but refers to another claim for some of its requirements. In this way, the claim “depends” on another claim. A dependent claim incorporates all of the requirements of the claim to which it refers. The dependent claim then adds its own additional requirements. To determine what a dependent claim covers, it is necessary to look at both the dependent claim and any other claim to which it refers. A product or method that meets all of the requirements of both the dependent claim and the claims to which it refers is covered by that dependent claim and the independent claim.

Instruction No. 15

INFRINGEMENT GENERALLY

I will now instruct you how to decide whether or not Synopsys has infringed the ’376 patent. Infringement is assessed on a claim-by-claim basis. Therefore, there may be infringement as to one claim but no infringement as to another.

In this case, there are three possible ways that a claim may be infringed. The three types of infringement are called: (1) direct infringement; (2) active inducement; and (3) contributory infringement. Active inducement and contributory infringement are referred to as indirect infringement. There cannot be indirect infringement without someone else engaging in direct infringement. To prove indirect infringement, Mentor Graphics must also prove that Synopsys’ indirect infringement caused direct infringement. In this case, Mentor Graphics has alleged that Synopsys directly infringes the ’376 patent. In addition, Mentor Graphics has alleged that Synopsys’ customers directly infringe the ’376 patent, and Synopsys is liable for actively inducing or contributing to that direct infringement by Synopsys’ customers.

In order to prove infringement, Mentor Graphics must prove that the requirements for one or more of these types of infringement are met by a preponderance of the evidence, i.e., that it is more likely than not that all of the requirements of one or more of each of these types of infringement have been proved.

I will now explain each of these types of infringement in more detail.

Instruction No. 16

DIRECT INFRINGEMENT

In order to prove direct infringement of method claims 1, 24, 26, and 27, Mentor Graphics must prove by a preponderance of the evidence, i.e., that it is more likely than not, that Synopsys performed every step of the asserted claim and did so without the permission of Mentor Graphics after October 4, 2012.

In order to prove direct infringement of claim 28, Mentor Graphics must prove by a preponderance of the evidence, i.e., that it is more likely than not, that Synopsys made, used, sold, offered for sale within, or imported into the United States a product that meets all the limitations of claim 28.

The asserted patent claims use the term “comprising,” meaning that they are “open claims.” An open claim is infringed if every limitation of the claim is met, regardless of whether additional features are present in the infringing method or apparatus. For example, in an open method claim that recites steps 1, 2, and 3, a method that includes steps 1, 2, 3, and additional step 4 infringes. And in an open apparatus claim that recites elements A, B, and C, an apparatus incorporating elements A, B, C, and additional feature D infringes. It is, thus, irrelevant to the infringement analysis whether the ZeBu emulation systems perform additional steps or contain additional elements beyond those recited in the asserted patent claims.

Mentor Graphics must prove by a preponderance of the evidence that the use or sale of ZeBu emulation systems using flexible probes or value change probes meets every element of an asserted claim.

You must determine, separately for each asserted claim, whether or not there is infringement. There is one exception to this rule. If you find that a claim on which other claims depend is not infringed, there cannot be infringement of any dependent claim that refers directly or indirectly to that independent claim. On the other hand, if you find that an independent claim has been infringed, you must still decide, separately, whether the additional requirements of any claims that depend from the independent claim are met, thus, whether those claims have also been infringed.

Instruction No. 17

INDIRECT INFRINGEMENT—ACTIVE INDUCEMENT

Mentor Graphics alleges that Synopsys is liable for infringement by actively inducing its customers to directly infringe the ’376 patent. As with direct infringement, you must determine whether there has been active inducement on a claim-by-claim basis.

Synopsys is liable for active inducement of a claim only if Mentor Graphics proves by a preponderance of the evidence that the following occurred after October 4, 2012:

1) a Synopsys customer carried out acts that directly infringe an asserted claim of the ’376 patent;

2) Synopsys took action specifically intending to cause a customer to carry out the infringing acts;

3) Synopsys was aware of the ’376 patent; and

4) Synopsys knew that the acts, if taken, would constitute infringement of the ’376 patent, or believed there was a high probability that the acts, if taken, would constitute infringement of the ’376 patent, but deliberately avoided confirming that belief.

In order to establish inducement of infringement, it is not sufficient that Synopsys’ customers directly infringe the claim. Nor is it sufficient that Synopsys was aware of the act(s) by its customers that allegedly constitute direct infringement. Rather, in order to find inducement of infringement, you must find that Synopsys specifically intended its customers to infringe the ’376 patent or that Synopsys believed there was a high probability that its customers would infringe the ’376 patent, but deliberately avoided learning the infringing nature of its customers’ acts.

Instruction No. 18

Indirect Infringement—Contributory Infringement

You must determine whether Synopsys is liable for contributory infringement by contributing to the direct infringement of the ’376 patent by its customers. As with direct infringement, you must determine contributory infringement on a claim-by-claim basis. Synopsys is liable for contributory infringement of a claim if Mentor Graphics proves by a preponderance of the evidence, i.e., that it is more likely than not that each of the following requirements is met and that each occurred after October 4, 2012:

1) direct infringement of the ’376 patent by a Synopsys customer;
2) Synopsys sells, offers to sell, or imports within the United States a component of an emulation system that is used to practice and asserted claim of the ’376 patent;
3) the component constitutes a material part of the claimed invention;
4) the component has no substantial, non-infringing use; and
5) Synopsys is aware of the ’376 patent and knows that there is no substantial, non-infringing use for the component.

The component may be a hardware component or a software feature.

Mentor Graphics does not need to prove that the entire emulation system has no non-infringing uses—only that the hardware component or software feature is not a common component suitable for non-infringing use. Synopsys may not escape liability as a contributory infringer merely by embedding the component in a larger product with other additional, non-infringing features.

Instruction No. 19

DAMAGES—GENERALLY

If you find that Synopsys infringed any asserted claim of the ’376 patent, you must then consider what amount of damages to award to Mentor Graphics. I will now instruct you about the measure of damages. By instructing you on damages, I am not suggesting which party should win this case.

The damages you award must be adequate to compensate Mentor Graphics for the infringement. They are not meant to punish Synopsys. While Mentor Graphics is not required to prove its damages with mathematical precision, it must prove them with reasonable certainty. Mentor Graphics has the burden to persuade you of the amount of its damages by a preponderance of the evidence. Your damages award should put Mentor Graphics in approximately the same financial position that it would have been in had the infringement not occurred. You should not award damages that are remote or speculative.

There are different types of damages that Mentor Graphics may be entitled to recover. In this case, Mentor Graphics seeks lost profits and a reasonable royalty. Lost profits consist of any actual reduction in business profits Mentor Graphics suffered as a result of Synopsys’ infringement. A reasonable royalty is defined as the money amount Mentor Graphics and Synopsys would have agreed upon as a fee for use of the invention before infringement began.

I will give more detailed instructions regarding damages shortly. Note, however, that Mentor Graphics is entitled to at least a reasonable royalty for each sale of an infringing emulation system.

Instruction No. 20

LOST PROFITS—"BUT FOR" TEST

In this case, Mentor Graphics seeks to recover lost profits for some of Synopsys' sales of Synopsys' emulation systems and a reasonable royalty on the rest of Synopsys' sales.

To recover lost profits (as opposed to reasonable royalties), Mentor Graphics must show a causal relationship between the infringement and Mentor Graphics' loss of profit. In other words, Mentor Graphics must show that, but for the infringement by Synopsys, there is a reasonable probability that Mentor Graphics would have earned higher profits. To show this, Mentor Graphics must prove that, if there had been no infringement, it would have made some portion of the sales that Synopsys made of the infringing product.

Synopsys' Sales to Intel

Mentor Graphics is entitled to recover lost profits on Synopsys' sales to Intel of its ZeBu emulation systems on its "two-supplier market" theory if it establishes each of the following:

1) That there were only two acceptable, available alternatives in the Intel market during the damages period: Mentor Graphics' emulation system and Synopsys' allegedly infringing emulation system.

2) That Mentor Graphics had the manufacturing and marketing capacity to make any infringing sales actually made by Synopsys and for which Mentor Graphics seeks an award of lost profits—in other words, that Mentor Graphics was capable of satisfying the demand. (See Instruction No. 23.)

3) The amount of profit that Mentor Graphics would have made if Synopsys had not infringed. (See Instruction No. 24.)

If Mentor Graphics meets all of the above requirements, the burden shifts to Synopsys to show that Mentor Graphics would not have made some or all of the diverted sales "but for" the infringement. Synopsys may do so by showing, for example, any of the following:

1) That the Cadence emulation system was an acceptable, available, non-infringing alternative to Mentor Graphics' emulation system and Synopsys' infringing emulation system at Intel. (See Instruction No. 22)

2) That an FPGA prototype was an acceptable, available, non-infringing alternative to Mentor Graphics' emulation system and Synopsys' infringing emulation system at Intel.

3) That a software simulator was an acceptable, available, non-infringing alternative to Mentor Graphics' emulation system and Synopsys' infringing emulation system at Intel.

4) That Synopsys could have made available during the damages period an acceptable, non-infringing alternative to Mentor Graphics' emulation system and Synopsys' infringing emulation system.

Instruction No. 20

5) That Intel would have bought fewer or no emulation systems in place of those it bought from Synopsys.

If Synopsys is able to show any of the above, you may reduce the amount of lost profits awarded to Mentor Graphics accordingly. For example, if you determine that, without the ZeBu emulation system in the market, Intel would have bought some Cadence emulation systems, instead of buying only Mentor Graphics emulation systems, you should only award Mentor Graphics lost profits for the sales made by Synopsys that you believe Mentor Graphics would actually have made to Intel. Or if you determine that, without the ZeBu emulation system in the market, Intel would have bought fewer or no Mentor Graphics emulation systems, you should only award Mentor Graphics lost profits for the sales made by Synopsys that you believe Mentor Graphics would actually have made to Intel.

Synopsys' Sales to Non-Intel Customers

Mentor Graphics is entitled to recover lost profits on Synopsys' sales to non-Intel customers of its ZeBu emulation systems if it establishes each of the following:

1) That there was a demand for the patented product. (See Instruction No. 21.)

2) Mentor Graphics' share of the non-Intel market. (See Instructions No. 25.)

3) That Mentor Graphics had the manufacturing and marketing capacity to make any infringing sales actually made by Synopsys and for which Mentor Graphics seeks an award of lost profits—in other words, that Mentor Graphics was capable of satisfying the demand. (See Instruction No. 23.)

4) The amount of profit that Mentor Graphics would have made if Synopsys had not infringed. (See Instruction No. 24.)

If Mentor Graphics meets all of the above requirements, the burden shifts to Synopsys to show that Mentor Graphics reasonably would not have made some or all of the diverted sales "but for" the infringement. Synopsys may do so by showing, for example, any of the following:

1) That Synopsys could have made available during the damages period an acceptable, non-infringing alternative to Mentor Graphics' emulation system and Synopsys' infringing emulation system.

2) That non-Intel customers would have bought fewer or no emulation systems in place of those it bought from Synopsys.

If Synopsys is able to show any of the above, you may reduce the amount of lost profits awarded to Mentor Graphics accordingly. For example, if you determine that, without the ZeBu emulation system in the market, non-Intel customers would have bought fewer or no Mentor Graphics emulation systems, you should only award Mentor Graphics lost profits for the sales made by Synopsys that you believe Mentor Graphics would actually have made to non-Intel customers.

Instruction No. 21

LOST PROFITS—DEMAND

With respect to the proving lost profits, demand for the patented product can be proven by:

1) significant sales of the patented product by Mentor Graphics, or

2) significant sales of an infringing Synopsys emulation system.

Instruction No. 22

LOST PROFITS—NON-INFRINGING SUBSTITUTES—ACCEPTABILITY

To be an "acceptable substitute," a product must have the advantages of the patented invention that were important to Intel. If Intel was motivated to buy Synopsys' infringing emulation system because of features and characteristics available only from Synopsys' infringing emulation system and Mentor Graphics' emulation system, then some other, alternative product that lacks the accused features is not an acceptable substitute, even if it otherwise competed with Synopsys' infringing emulation system and Mentor Graphics' emulation system. On the other hand, if the realities of the marketplace are that competitors other than Mentor Graphics would likely have captured the sales made by Synopsys, despite a difference in the products, then Mentor Graphics is not entitled to lost profits on those sales.

An alternative product may be considered "available" as a potential substitute even if the product was not actually on sale during the infringement period. Factors suggesting the alternative was available include whether the material, experience, and know-how for the alleged substitute were readily available at the time of infringement. Factors suggesting the alternative was not available include whether the material was of such high cost as to render the alternative unavailable and whether Synopsys had to design or invent around the patented technology to develop an alleged substitute.

Instruction No. 23

LOST PROFITS—CAPACITY

With respect to the third requirement for lost profits, a patent holder is only entitled to lost profits for sales it could have made. In other words, Mentor Graphics must show that it had the manufacturing and marketing capability to make the sales it said it lost: Mentor Graphics must show it is more probable than not that it could have made and sold, or could have had someone else make or sell for it, the additional products it says it could have sold but for the infringement.

Instruction No. 24

Lost Profits—Amount of Profit

With respect to the last requirement for lost profits, Mentor Graphics may calculate its lost profits on lost sales by computing the lost revenue for sales it has proven it would have made but for the infringement and subtracting from that figure the amount of additional costs or expenses it would have incurred in making those lost sales, such as cost of goods, sales costs, packaging costs, and shipping costs. Certain fixed costs that do not vary with increases in production or scale, such as taxes, insurance, rent, and administrative overhead, should not be subtracted from a patent holder's lost revenue.

Instruction No. 25

NON-INTEL LOST PROFITS—MARKET SHARE

If Mentor Graphics establishes it would have made some, but not all, of Synopsys' sales but for the infringement, the amount of sales that Mentor Graphics lost may be shown by proving Mentor Graphics' share of the relevant market, excluding infringing products. Mentor Graphics may be awarded a share of profits equal to its market share even if there were non-infringing substitutes available. In determining Mentor Graphics' market share, the market must be established first, which requires determining which products are in that market. Products are considered in the same market if they are considered "sufficiently similar" to compete against each other. Two products may be sufficiently similar if they do not have significantly different prices or characteristics.

Instruction No. 26

Lost Profits—Collateral Sales

In this case, Mentor Graphics is seeking lost profits from sales of service agreements, which Mentor Graphics contends it would have sold along with its emulation system that competes with Synopsys' emulation systems. These service agreements sold with Mentor Graphics' emulation systems are called collateral products.

To recover lost profits on sales of such collateral products, Mentor Graphics must establish it is more likely than not that Mentor Graphics would have sold the collateral products (the service agreements) but for the infringement.

Recovery for lost profits on sales of collateral products must not include items that essentially have no functional relationship to the competitive product and that have been sold with the competitive product only as a matter of convenience or business advantage. You may not award patent damages for sales of items or services that are neither competitive with nor function with the patented invention.

Instruction No. 27

REASONABLE ROYALTY—GENERALLY

If you find that Mentor Graphics has established infringement of the Mentor Graphic patent, Mentor Graphics is entitled to at least a reasonable royalty to compensate it for that infringement. If you find that Mentor Graphics has not proved its claim for lost profits, or has proved its claim for lost profits for only a portion of the infringing sales, then you must award Mentor Graphics a reasonable royalty for all infringing sales for which you have not awarded lost profits damages.

Instruction No. 28

REASONABLE ROYALTY—DEFINITION

A royalty is a payment made to a patent holder in exchange for the right to make, use, or sell the claimed invention. A reasonable royalty is the amount of royalty payment that a patent holder and the infringer would have agreed to in a hypothetical negotiation taking place at a time prior to when the infringement first began. In considering this hypothetical negotiation, you should focus on what Mentor Graphics' and Synopsys' expectations would have been had they entered into an agreement when infringement first began, and had they acted reasonably in their negotiations. In determining this, you must assume that both parties believed the patent was infringed and that Mentor Graphics and Synopsys were willing to enter into an agreement. The reasonable royalty you determine must be a royalty that would have resulted from the hypothetical negotiation, and not simply a royalty either party would have preferred. Evidence of things that happened after the infringement first began can be considered in evaluating the reasonable royalty only to the extent that the evidence aids in assessing what royalty would have resulted from a hypothetical negotiation. Although evidence of the actual profits Synopsys made may be used to determine the anticipated profits at the time of the hypothetical negotiation, the royalty may not be limited or increased based on the actual profits that Synopsys made.

Instruction No. 29

REASONABLE ROYALTY—RELEVANT FACTORS

In determining the reasonable royalty, you should consider all the facts known and available to the parties at the time the infringement began. Some of the kinds of factors that you may consider in making your determination are:

1) The royalties received by Mentor Graphics for the licensing of the ’376 patent, proving or tending to prove an established royalty.

2) The rates paid by Synopsys for the use of other patents comparable to the ’376 patent.

3) The nature and scope of the license, as exclusive or nonexclusive, or as restricted or nonrestricted in terms of territory or with respect to whom the manufactured product may be sold.

4) Mentor Graphics’ established policy and marketing program to maintain its patent monopoly by not licensing others to use the invention or by granting licenses under special conditions designed to preserve that monopoly.

5) The commercial relationship between Mentor Graphics and Synopsys, such as whether they are competitors in the same territory in the same line of business, or whether they are inventor and promoter.

6) The effect of selling the patented feature in promoting sales of other Synopsys products, the existing value of the invention to Mentor Graphics as a generator of sales of its nonpatented items, and the extent of such derivative or collateral sales.

7) The duration of the ’376 patent and the term of the license.

8) The established profitability of Synopsys’ and Mentor Graphics’ products covered by the ’376 patent, their commercial success, and current popularity.

9) The utility and advantages of the patented feature over the old modes or devices, if any, that had been used for working out similar results.

10) The nature of the patented invention, the character of the commercial embodiment of it as owned and produced by the licensor, and the benefits to those who have used the invention.

11) The extent to which Synopsys has made use of Mentor Graphics’ invention and any evidence probative of the value of that use.

12) The portion of the profit or of the selling price that may be customary in the particular business or in comparable business to allow for the use of the invention or analogous inventions.

Instruction No. 29

13) The portion of the realizable profits that should be credited to Mentor Graphics' patented features as distinguished from nonpatented features, the manufacturing process, business risks, or significant features or improvements added by Synopsys.

14) The opinion and testimony of qualified experts.

15) The amount that a licensor, such as Mentor Graphics, and a licensee, such as the Synopsys, would have agreed upon at the time the infringement began if both had been reasonably and voluntarily trying to reach an agreement; that is, the amount which a prudent licensee—who desired, as a business proposition, to obtain a license to manufacture and sell a particular article embodying the patented invention—would have been willing to pay as a royalty and yet be able to make a reasonable profit and which amount would have been acceptable by a prudent patentee who was willing to grant a license.

No one factor is dispositive and you can and should consider the evidence that has been presented to you in this case on each of these factors. You may also consider any other factors which in your mind would have increased or decreased the royalty Synopsys would have been willing to pay and Mentor Graphics would have been willing to accept, acting as normally prudent business people. The final factor establishes the framework which you should use in determining a reasonable royalty, that is, the payment that would have resulted from a negotiation between Mentor Graphics and Synopsys taking before Synopsys' alleged infringement began.

Instruction No. 30

DATE OF COMMENCEMENT OF DAMAGES—PRODUCTS

In determining the amount of Mentor Graphics' damages, you should assume that damages begin on October 4, 2012.

Instruction No. 31

USE OF NOTES

Some of you have taken notes during the trial. Whether or not you took notes, you should rely on your own memory of what was said. Notes are only to assist your memory. You should not be overly influenced by your notes or those of your fellow jurors.

Instruction No. 32

COMMUNICATION WITH COURT

If it becomes necessary during your deliberations to communicate with me, you may send a note through the clerk signed by your presiding juror or by one or more members of the jury. No member of the jury should ever attempt to communicate with me except by a signed writing. I will communicate with any member of the jury on anything concerning the case only in writing, or here in open court. If you send out a question, I will consult with the parties before answering it, which may take some time. You may continue your deliberations while waiting for the answer to any question. Remember that you are not to tell anyone—including me—how the jury stands, numerically or otherwise, until after you have reached a unanimous verdict or have been discharged. Do not disclose any vote count in any note to the court.

Instruction No. 33

RETURN OF VERDICT

A verdict form has been prepared for you. After you have reached unanimous agreement on a verdict, your presiding juror will fill in the form that has been given to you, sign and date it, and advise the court that you are ready to return to the courtroom.

Instruction No. 34

GLOSSARY

Some of the terms in this glossary will be defined in more detail in the legal instructions you are given. The definitions in the instructions must be followed and must control your deliberations.

Abstract: A brief summary of the technical disclosure in a patent to enable the U.S. Patent and Trademark Office and the public to determine quickly the nature and gist of the technical disclosure in the patent.

Amendment: A patent applicant's change to one or more claims or to the specification either in response to an office action taken by an Examiner or independently by the patent applicant during the patent application examination process.

Assignment: A transfer of patent rights to another called an "assignee" who, upon transfer, becomes the owner of the rights assigned.

Claim: Each claim of a patent is a concise, formal definition of an invention and appears at the end of the specification in a separately numbered paragraph. In concept, a patent claim marks the boundaries of the patent in the same way that a legal description in a deed specifies the boundaries of land, i.e., similar to a landowner who can prevent others from trespassing on the bounded property, the inventor can prevent others from using what is claimed. Claims may be independent or dependent. An independent claim stands alone. A dependent claim does not stand alone and refers to one or more other claims. A dependent claim incorporates whatever the other referenced claim or claims say.

Drawings: The drawings are visual representations of the claimed invention contained in a patent application and issued patent, and usually include several figures illustrating various aspects of the claimed invention.

Elements: The required parts of a device or the required steps of a method. A device or method infringes a patent if it contains each and every requirement of a patent claim.

Embodiment: A product or method that contains the claimed invention.

Examination: Procedure before the U.S. Patent and Trademark Office whereby an Examiner reviews the filed patent application to determine if the claimed invention is patentable.

Infringement: Violation of a patent occurring when someone makes, uses, or sells a patented invention, without permission of the patent holder, within the United States during the term of the patent. Infringement may be direct, by inducement, or contributory. Direct infringement is making, using, or selling the patented invention without permission. Inducing infringement is intentionally causing another to directly infringe a patent. Contributory infringement is offering to sell or selling an item that is a significant part of the invention, so that the buyer directly infringes the patent. To be a contributory infringer, one must know that the part being offered or sold is designed specifically for infringing the patented invention and is not a common object suitable for non-infringing uses.

Instruction No. 34

Limitation: A required part of an invention set forth in a patent claim. A limitation is a requirement of the invention. The word “limitation” is often used interchangeably with the word "requirement.”

Patent: A patent is an exclusive right granted by the U.S. Patent and Trademark Office to an inventor to prevent others from making, using, or selling an invention for a term of 20 years from the date the patent application was filed (or 17 years from the date the patent issued). When the patent expires, the right to make, use, or sell the invention is dedicated to the public. The patent has three parts, which are a specification, drawings and claims. The patent is granted after examination by the U.S. Patent and Trademark Office of a patent application filed by the inventor which has these parts, and this examination is called the prosecution history.

Patent and Trademark Office (PTO): An administrative branch of the U.S. Department of Commerce that is charged with overseeing and implementing the federal laws of patents and trademarks. It is responsible for examining all patent applications and issuing all patents in the United States.

Prior Art: Previously known subject matter in the field of a claimed invention for which a patent is being sought. It includes issued patents, publications, and knowledge deemed to be publicly available, such as trade skills, trade practices, and the like.

Prosecution History: The prosecution history is the complete written record of the proceedings in the PTO from the initial application to the issued patent. The prosecution history includes the office actions taken by the PTO and the amendments to the patent application filed by the applicant during the examination process.

Reads On: A patent claim “reads on” a device or method when each required part (requirement) of the claim is found in the device or method.

Requirement: A required part or step of an invention set forth in a patent claim. The word “requirement” is often used interchangeably with the word “limitation.”

Royalty: A royalty is a payment made to the owner of a patent by a nonowner in exchange for rights to make, use, or sell the claimed invention.

Specification: The specification is a required part of a patent application and an issued patent. It is a written description of the invention and of the manner and process of making and using the claimed invention.

Jury Verdict Form, Dated October 10, 2014 (DKT 723)

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF OREGON

PORTLAND DIVISION

MENTOR GRAPHICS CORPORATION, an Oregon Corporation,

Plaintiffs and Counter-Defendant,

v.

EVE-USA, INC., a Delaware corporation, and **SYNOPSYS EMULATION AND VERIFICATION S.A.,** formed under the laws of France,

Defendants and Counter-Claimants.

Case No. 3:10-cv-00954-MO (Lead)
Case No. 3:12-cv-01500-MO
Case No. 3:13-cv-00579-MO

VERDICT FORM

VERDICT FORM

When answering the following questions and filling out this Verdict Form, please follow the directions provided throughout the form. Your answer to each question must be unanimous. Some of the questions contain legal terms that are defined and explained in detail in the Jury Instructions. Please refer to the Jury Instructions if you are unsure about the meaning or usage of any legal term that appears in the questions below.

We, the jury, unanimously agree to the answers to the following questions and return them under the instructions of this Court as our verdict in this case.

FINDINGS ON INFRINGEMENT CLAIMS

I. Flexible Probes

A. Direct Infringement

1. Has Mentor Graphics proven that it is more likely than not that Synopsys' use, sale, or importation of emulators using flexible probes infringes one or more asserted claims of the Mentor Graphics patent?

Claim	Verdict on direct infringement
1	[✓ yes] no
24	[✓ yes] no
26	[✓ yes] no
27	[✓ yes] no
28	[✓ yes] no

B. Inducing Infringement

2. Has Mentor Graphics proven that it is more likely than not that Synopsys induced infringement of one or more claims of the Mentor Graphics patent through Synopsys' use, sale, or importation of emulators using flexible probes?

Claim	Verdict on induced infringement
1	[✓ yes] no
24	[✓ yes] no
26	[✓ yes] no
27	[✓ yes] no
28	[✓ yes] no

C. Contributory Infringement

3. Has Mentor Graphics proven that it is more likely than not that Synopsys contributed to the infringement of one or more claims of the Mentor Graphics patent Synopsys' use, sale, or importation of emulators using flexible probes?

Claim	Verdict on contributory infringement
1	[✓ yes] ☐ no
24	[✓ yes] ☐ no
26	[✓ yes] ☐ no
27	[✓ yes] ☐ no
28	[✓ yes] ☐ no

II. Value Change Probes

A. Direct Infringement

4. Has Mentor Graphics proven that it is more likely than not that Synopsys' use, sale, or importation of emulators using value change probes infringes one or more asserted claims of the Mentor Graphics patent?

Claim	Verdict on direct infringement
1	[✓ yes] ☐ no
24	[✓ yes] ☐ no
26	[✓ yes] ☐ no
27	[✓ yes] ☐ no
28	[✓ yes] ☐ no

B. Inducing Infringement

5. Has Mentor Graphics proven that it is more likely than not that Synopsys induced infringement of one or more claims of the Mentor Graphics patent through Synopsys' use, sale, or importation of emulators using value change probes?

Claim	Verdict on induced infringement
1	[✓ yes] ☐ no
24	[✓ yes] ☐ no
26	[✓ yes] ☐ no
27	[✓ yes] ☐ no
28	[✓ yes] ☐ no

C. Contributory Infringement

6. Has Mentor Graphics proven that it is more likely than not that Synopsys contributed to the infringement of one or more claims of the Mentor Graphics patent through Synopsys' use, sale, or importation of emulators using value change probes?

Claim	Verdict on contributory infringement
1	[✓ yes] ☐ no
24	[✓ yes] ☐ no
26	[✓ yes] ☐ no
27	[✓ yes] ☐ no
28	[✓ yes] ☐ no

FINDINGS ON DAMAGES (IF APPLICABLE)

If you answered question 1, 2, 3, 4, 5and/or 6 "yes," proceed to answer the remaining questions. If you did not so answer, do not answer the remaining questions and proceed to check and sign the verdict form.

4. What lost profits, if any, did Mentor Graphics show it more likely than not suffered as a result of sales that it would have made but for Synopsys' infringement?

$ 36,417,661.00

5. For those infringing sales for which Mentor Graphics has not proved its entitlement to lost profits, what has it proved it is entitled to as a reasonable royalty:

a) royalty payment of 5 % of $ 4,842,209.00 in total sales;

OR

b) one-time payment of $ ______________

You have now reached the end of the verdict form and should review it to ensure it accurately reflects your unanimous determinations. The Presiding Juror should then sign and date the verdict form in the spaces below and notify the Security Guard that you have reached a verdict. The Presiding Juror should retain possession of the verdict form and bring it when the jury is brought back into the courtroom.

DATED: October 10, 2014

By: [redacted]
Presiding Juror

Final Judgment, Entered November 17, 2014 (DKT 769)

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF OREGON

PORTLAND DIVISION

MENTOR GRAPHICS CORPORATION, an Oregon Corporation,
Plaintiff and Counter-Defendant,
v.
EVE-USA, INC., a Delaware corporation, and **SYNOPSYS EMULATION AND VERIFICATION S.A.,** formed under the laws of France,
Defendants and Counter-Claimants.

SYNOPSYS, INC., a Delaware Corporation, **EVE-USA, INC.,** a Delaware corporation, and **SYNOPSYS EMULATION AND VERIFICATION S.A.,** formed under the laws of France,
Plaintiffs and Counter-Defendants,
v.
MENTOR GRAPHICS CORPORATION, an Oregon Corporation.
Defendant and Counter-Claimant.

Case No. 3:10-cv-00954-MO (Lead)
Case No. 3:12-cv-01500-MO
Case No. 3:13-cv-00579-MO

FINAL JUDGMENT

In these consolidated actions, Mentor Graphics Corporation ("Mentor Graphics") asserted five counts of patent infringement: infringement of U.S. Patent Nos. 6,876,962 ("the '962 patent"); 6,947,882 ("the '882 patent"); 6,240,376 ("the '376 patent"); 6,009,531 ("the '531 patent"); and 5,649,176 ("the '176 patent"). Synopsys, Inc., EVE-USA, Inc. and Synopsys Emulation and Verification S.A. ("Synopsys") asserted two counts for declaratory relief that the '376, '531, and '176 patents were invalid and not infringed, and Synopsys, Inc. asserted two counts of infringement, namely infringement of U.S. Patent Nos. 7,069,526 ("the '526 patent") and 6,132,109 ("the '109 patent").

This action came before the Court through various summary judgment motions and a jury trial, and was resolved as follows:

U.S. Patent No. 6,240,376

As set forth in the Court's February 21, 2014 summary judgment order, judgment is entered in favor of Mentor Graphics on Synopsys' declaratory relief claim that the '376 patent is invalid on the ground that assignor estoppel bars Synopsys from challenging the validity of the '376 patent. (Dkt. 472.)

On October 10, 2014, the jury rendered its verdict. With respect to and in accordance with that verdict, judgment is entered in favor of Mentor Graphics as follows:

1. Synopsys' use, sale or importation of ZeBu emulators using flexible probes and value change probes, in the manner set forth by Mentor Graphics at trial, directly infringes claims 1, 24, 26, 27, and 28 of U.S. Patent No. 6,240,376.
2. Synopsys' use, sale or importation of ZeBu emulators using flexible probes and value change probes, in the manner set forth by Mentor Graphics at trial, induces infringement of claims 1, 24, 26, 27, and 28 of U.S. Patent No. 6,240,376.
3. Synopsys' use, sale or importation of ZeBu emulators using flexible probes and value change probes, in the manner set forth by Mentor Graphics at trial, contributes to the infringement of claims 1, 24, 26, 27, and 28 of U.S. Patent No. 6,240,376.
4. Mentor Graphics is awarded $36,417,661.00 in lost profits.

5. Mentor Graphics is also awarded royalty payments of $242,110.45, which is 5% of $4,842,209.00.

(Dkt. 723.)

U.S. Patent No. 5,649,176

As set forth in the Court's June 4, 2014 summary judgment order, judgment is entered in favor of Synopsys on Mentor Graphics' claims for infringement of the '176 patent on the ground that the doctrine of claim preclusion bars such claims. (Dkt. 524.) As set forth in the parties' Stipulation to Dismiss Without Prejudice Synopsys' Claims for Declarations of Invalidity and Non-Infringement of U.S. Patent Nos. 5,649,176 and 6,009,531, Synopsys' claim for declaratory judgment of invalidity and non-infringement of the '176 patent has been dismissed without prejudice. (Dkt. 729.)

U.S. Patent No. 6,009,531

As set forth in the Court's June 4, 2014 summary judgment order, judgment is entered in favor of Synopsys on Mentor Graphics' claims for infringement of the '531 patent on the ground that the doctrine of claim preclusion bars such claims. (Dkt. 524.) As set forth in the parties' Stipulation to Dismiss Without Prejudice Synopsys' Claims for Declarations of Invalidity and Non-Infringement of U.S. Patent Nos. 5,649,176 and 6,009,531, Synopsys' claim for declaratory judgment of invalidity and non-infringement of the '531 patent has been dismissed without prejudice. (Dkt. 729.)

U.S. Patent No. 6,132,109

As set forth in the Court's July 25, 2014 summary judgment order, judgment is entered in favor of Mentor Graphics on Synopsys' claims for infringement of the '109 patent on the ground that Claim 1 is invalid. (Dkt. 581).

U.S. Patent No. 7,069,526

As set forth in the Court's July 25, 2014 summary judgment order, judgment is entered in favor of Mentor Graphics on Synopsys' claims for infringement of the '526 patent on the grounds that Claims 19, 24, 28, 30, and 33 are invalid. (Dkt. 581).

U.S. Patent No. 6,876,962

As set forth in the Court's July 29, 2014 summary judgment order, judgment is entered in favor of Synopsys on Mentor Graphics' claim for infringement of the '962 patent on the grounds that Claims 3, 5, 6, and 8 are not infringed. (Dkt. 582).

U.S. Patent No. 6,947,882

As set forth in the Court's July 25, 2014 summary judgment order, judgment is entered in favor of Synopsys on Mentor Graphics claim for infringement of the '882 Patent on the grounds that Claims 2 and 4 are not infringed. (Dkt. 581.) As set forth in the Court's July 29, 2014 summary judgment order, judgment is entered in favor of Synopsys on Mentor Graphics' claim for infringement of the '882 patent on the grounds that Claims 7, 9, and 13 are invalid. (Dkt. 582.)

Therefore, IT IS HEREBY ORDERED AND ADJUDGED that judgment is entered in favor of Synopsys on Mentor Graphics' claims for infringement of the '962, '882, '531, and '176 patents. Synopsys' claims for a declaration that the '531 and '176 patents are invalid and not infringed have been dismissed without prejudice. It is further ORDERED AND ADJUDGED that judgment be entered in favor of Mentor Graphics on Mentor Graphics' claim for infringement of the '376 patent, on Synopsys' claims for infringement of the '109 and '526 patents, and on Synopsys' claim for a declaration that the '376 patent is not infringed and is invalid. Mentor Graphics is awarded $36,417,661.00 in lost profits damages and a royalty payment of 5% of $4,842,209.00, which is $242,110.45.

The parties agree that each party shall bear its own costs, and neither party shall seek to recover attorneys' fees arising from any claims or causes of action resolved by this final judgment.

DATED: November 17, 2014

/s/ Michael W. Mosman
MICHAEL W. MOSMAN
United States District Judge

Opinion and Order, Dated March 11, 2015 (DKT 842)

UNITED STATES DISTRICT COURT

DISTRICT OF OREGON

PORTLAND DIVISION

MENTOR GRAPHICS CORPORATION,
an Oregon Corporation,

Plaintiff/Counter-defendant,

v.

EVE-USA, INC., a Delaware corporation; and **SYNOPSYS EMULATION AND VERIFICATION S.A.,** formed under the laws of France,

Defendants/Counter-claimants.

EVE-USA, INC., a Delaware corporation; and **SYNOPSYS EMULATION AND VERIFICATION S.A.**, formed under the laws of France,

Plaintiffs/Counter-defendants

v.

MENTOR GRAPHICS CORPORATION,
an Oregon corporation,

Defendant/Counter-claimant.

Case No. 3:10-cv-954-MO (lead)
Case No. 3:12-cv-1500-MO
Case No. 3:13-cv-579-MO

OPINION AND ORDER

MOSMAN, J.,

On October 10, 2014, a jury entered a verdict finding that Defendants (collectively "Synopsys") were liable for direct and contributory infringement of Plaintiff's ("Mentor") patent. Verdict [722] at 1–3. The jury awarded Mentor $36,417,661 for lost profits from sales Mentor proved it would have made but for the infringement and 5% of $4,842,209 as a reasonable royalty for sales that Mentor failed to prove it would have made but for the infringement. *Id.* at 3. Since the entry of the jury verdict, Mentor filed a Motion for Accounting [783] and Synopsys filed a Motion for Judgment as a Matter of Law ("JMOL") [787] and a Motion for New Trial on Damages [790].

On March 3, 2015, an oral argument was held regarding these motions. This opinion and order will dispose of all the issues taken under advisement at the close of oral argument.

I. Motion for Accounting

Mentor's Motion for Accounting [783] is DENIED. For the reasons stated on the record at oral argument, I order that a new trial be held to determine the amount of supplemental damages Mentor is entitled to receive. The two remaining sets of issues after oral argument were: (1) whether a jury trial would be necessary to determine the amount of supplemental damages Mentor is entitled to, and whether that trial would result in what I called a jury trial trap; and (2) whether Mentor would be able to seek pre-verdict supplemental damages at this new trial. For the following reasons, I do not believe this order will result in a jury trial trap, and I believe that Mentor has the right to seek pre-verdict supplemental damages.

A. *Jury Trial Issues*

1. The Right to a Jury Trial

At oral argument and in its reply brief, Mentor argued that I had the discretion to decide how to determine the amount of supplemental damages Mentor was entitled to; a jury trial was not required. In addition, Mentor raised the issue of a potential jury trial trap if I were to order a new jury trial to determine the supplemental damages at issue. In other words, there would be a trap because using a jury trial would trigger a line of Federal Circuit opinions regarding res judicata that in this context would require me to dismiss the supplemental damages trial. I believe a jury trial is required to determine the amount of supplemental damages Mentor is entitled to, but I do not believe it is appropriate to determine the potential jury trial trap issue at this time.

At oral argument and in its briefing, Mentor argued that I was not required to hold a jury trial to determine supplemental damages. Mentor relied on *SynQor, Inc. v. Artesyn Technologies, Inc.*, in which the Federal Circuit held:

> [T]he amount of supplemental damages following a jury verdict is a matter committed to the sound discretion of the district court . . . . [A] jury right is not implicated every time the district court is required to determine factual matters before awarding supplemental damages to compensate the patentee for post-verdict infringement.

709 F.3d 1365, 1384 (Fed. Cir. 2013) (internal quotations omitted). I believe Mentor's reliance on this case is misplaced. The fact that the Federal Circuit said a jury right is not implicated *every time* when it could have said that a jury right is *never* implicated indicates there are circumstances where determining certain factual matters would trigger a jury right. I believe this to be such a case. In *Apple, Inc. v. Samsung Electronics Co., Ltd.*, the district court held it could award supplemental lost profits damages, but not royalty damages, without triggering a jury trial

right. 926 F. Supp. 2d 1100, 1106 (N.D. Cal 2013). With respect to supplemental lost profits damages, the court held, "Because the jury returned an award for each product separately, the Court can simply divide the jury award for each product by that product's number of sales to calculate [the lost profits] per-product amount." *Id.* With respect to supplemental royalty damages, the court held, "Here, the jury did not make a finding as to the appropriate royalty rate, and the Court cannot now do so without treading on Samsung's Seventh Amendment right to a jury trial on that issue." *Id.* In other words, where an award for supplemental damages would require the district court to engage in additional fact-finding of essential facts to the proffered damages theory, a jury right is triggered. In *Apple*, the court was able to award lost profits because no additional fact-finding was required. The district court was able to take the per-product lost profits amount found by the jury and apply it to any undisputed sales not considered by the jury. However, the district court could not award supplemental royalty damages because that would have required the district court to determine the appropriate royalty rate, i.e. an essential fact to a royalty award not determined by the jury.

In our case, in order to extrapolate an award of supplemental lost profits from the jury award I would need to find that the Intel two-supplier market continued after December 31, 2013. An Intel two-supplier market is an essential fact to Mentor's damages theory, but the jury was not required to determine, nor given any evidence to determine, whether the two-supplier market continued after December 31, 2013. Synopsys argues that sometime in 2014, Intel acquired a small tech company called Avago, which had a Palladium emulator. Defs.' Response in Opposition [808] at 3–4. Synopsys argues that it can prove that the Palladium met Intel's needs and that historical precedent would show that once an emulation provider has a foothold within a company, such as the Palladium emulator in this case, it is likely to grow its presence.

Id. This would undermine an essential factual basis of the jury award—i.e. a two-supplier Intel market. Based on the principal found in *Apple*, this alleged factual change triggers a jury trial right.

2. Jury Trial Trap

Mentor argues that if I grant Synopsys's request for a jury trial to determine supplemental damages, Synopsys will then argue that a line of Federal Circuit cases bars the trial I have just ordered—this is what I called the jury trial trap at oral argument. Pl.'s Reply [816] at 8. Although I expressed some tentative views on this issue at oral argument, I do not believe this issue is ripe. Mentor may be correct that Synopsys will raise this argument when Mentor moves for a trial on supplemental damages, but because Mentor is yet to move for a trial and because Synopsys has not yet made this argument, it is too early for me rule on this issue. Whether or not there is a jury trial trap does nothing to change the fact that Synopsys's Seventh Amendment rights require that there be a trial. I leave it to a later summary judgment motion to decide whether or not a jury trial is barred by res judicata.

B. *Pre-Verdict Supplemental Damages*

District courts have the authority to award supplemental damages based on pre-verdict infringement not considered by the jury. *See Metso Minerals, Inc. v. Powerscreen Int'l Distrib. Ltd.*, 833 F. Supp. 2d 333, 351 (E.D. N.Y. 2011) (awarding supplemental damages for pre-verdict infringement not considered by jury); *ActiveVideo Networks, Inc. v. Verizon Commc'ns., Inc.*, No. 10 Civ. 248, 2011 WL 4899922, at *4 (E.D. Va. Oct. 14, 2011) (awarding supplemental damages for the approximate five month period predating the trial and jury verdict); *Hynix Semiconductor, Inc. v. Rambus, Inc.*, 609 F.Supp.2d 951, 959–60, 987 (N.D. Cal. 2009) (awarding pre-verdict supplemental damages after "the last date for which [the patentee] was able to present evidence of [infringing] sales to the jury"); *Itron, Inc. v. Benghiat*, No. CIV.99-501 (JRT/FLN), 2003 WL 22037710, at *15–16 (D. Minn. Aug. 29, 2003) (awarding damages for pre-verdict period of infringement for which infringer provided no sales data); *Mikohn Gaming v. Acres Gaming, Inc.*, No. CV-S-97-1383-EJW, 2001 WL 34778689, at *19 (D. Nev. Aug. 2, 2001) (awarding supplemental damages that include pre-verdict infringing sales not contained in the damages experts' reports nor presented to the jury). However, Synopsys correctly cites *Oscar Mayer Food Corporation v. Conagra, Inc.* for the proposition that "[if] it is not clear whether the jury awarded damages for the period of time up to and including the date of trial," the "awarding [of] additional amounts for damages" would improperly invade "the jury's province to determine actual damages." 869 F. Supp. 656, 668 (W.D. Wis. 1994). Synopsys argued that it is not clear whether the jury awarded damages for the pre-verdict period starting with the close of discovery and running up to when the verdict was entered because: (1) the verdict form and instructions did not contain a damages cutoff date; and (2) a jury question submitted just prior to the jury entering a verdict. I do not find these arguments to be sufficiently persuasive to justify barring Mentor from seeking pre-verdict supplemental damages.

In *Telecordia Technologies, Inc. v. Cisco Systems, Inc.*, the Federal Circuit held that "[d]istrict courts have broad discretion to interpret an ambiguous verdict form, because district courts witness and participate directly in the jury trial process. The district court was in a position to assess whether the verdict figure represented past infringement as well as ongoing infringement." 612 F.3d 1365, 1378 (Fed. Cir. 2010). Although the verdict form and jury instructions did not contain an explicit damages cutoff date, based on the evidence presented at trial, I believe the jury did not consider or award damages for the period of time between the close of discovery (December 31, 2013) and the entry of the verdict (October 10, 2014). First, as

Mentor points out in its briefing, both sides' experts limited their damages calculations to sales and market conditions up to and including December 31, 2013. Neither expert included models or a proposed methodology for how those calculations could be projected into the future. Second, I do not view the jury note as strong evidence that the jury awarded damages for any period of time after December 31, 2013. The jury question stated, "Where in the binders can we find Eve sales to Intel from 2006 to present." Jury Question [724]. Although the question asks for data up to the "present," this could simply have been imprecise language from the jury. It is not unreasonable to think that when the jury asked for data up to the present, it meant up until 2013—the cutoff for the relevant time period it heard about at trial. What is more telling, is that the jury only asked for sales data from Eve to Intel, and not Eve to all customers. If the jury had intended to award damages for the period of time after December 31, 2013 through the verdict, it should have been asking for all Eve sales data regardless of the customer. Just as the jury had not received any sales data for Eve to Intel post-2013—which it apparently looked for and could not find—it had not received any sales data of Eve to other companies, and therefore would not have been able to find all the necessary Eve sales data had it tried to award damages for that period. Although the verdict form and jury instructions could have been clearer, based on the evidence presented at trial, it is clear that the jury did not award damages for any period of time after December 31, 2013. Mentor is therefore entitled to seek pre-verdict supplemental damages.

II. Motion for Judgment as a Matter of Law

Synopsys's Motion for JMOL [787] is GRANTED in part and DENIED in part. For the reasons stated on the record, I reject Synopsys's arguments that: (1) Mentor failed to prove direct infringement; (2) Mentor failed to apportion its damages; (3) Mentor failed to prove that the relevant market was inelastic; (4) Mentor failed to prove Intel was in a two-supplier market; and

(5) Mentor failed to properly support its reasonable royalty arguments. The two remaining issues after oral argument were: (1) whether Mentor in fact made a prima facie case of contributory infringement; and (2) whether Mentor's use of the Veloce Quattro in its damages calculation was proper. For the following reasons, I do not believe Mentor established a prima facie case of contributory infringement, but I do believe it was proper for Mentor to use the Veloce Quattro in its damages calculations. Synopsys's JMOL motion is therefore granted with respect to its contributory infringement arguments, but denied in all other regards.

A. *Contributory Infringement*

In order to establish a claim for contributory infringement, a plaintiff must show that the accused material or apparatus used in the patented process lacks a substantial non-infringing use. *In Re Bill of Lading Transmission*, 681 F.3d 1323, 1337 (Fed. Cir. 2012) (internal quotation marks and citations omitted). "For purposes of contributory infringement, the inquiry focuses on whether the accused products can be used for purposes *other than* infringement." *Id.* at 1338 (emphasis in original). "Where the product is equally capable of, and interchangeably capable of both infringing and substantial non-infringing uses, a claim for contributory infringement does not lie." *Id.* Based on the evidence presented at trial, in the briefing, and at oral argument, I agree with Synopsys that Mentor failed to prove that flexible and value change probes lacked a substantial non-infringing use. In its response, Mentor states, "[We] made [our] prima facie case by demonstrating that the flexible and value change probes practice the '376 Patent." Pl.'s Response in Opposition [809] at 21. Merely showing infringement, however, is not sufficient to establish a claim for contributory infringement; Mentor also had to show that flexible and value change probes lacked any substantial non-infringing use. Mentor has failed to point to any evidence in the record presented at trial from which the jury could reasonably have concluded that flexible and value change probes lacked a substantial non-infringing use. I therefore grant Synopsys Motion for JMOL with regards to Mentor's contributory infringement claim.

B. ***Use of the Veloce Quattro in Damages Calculation***

Synopsys argued that Mentor's use of the Veloce Quattro in its damages presentation fatally flawed its damages theory because the evidence showed the Veloce Quattro was not suitable for Intel. Synopsys argues that because Mentor failed to prove it had a product Intel would have actually bought had ZeBu been pulled off the shelf, Mentor failed to prove that but for Synopsys's infringement of the '376 Patent it would have made any additional profits.

Synopsys's argument that there was extensive evidence that the Intel processor group would not buy the Veloce Quattro in place of the ZeBu has no basis. Defs.' Reply [819] at 28–29. The fact that the Intel processor group had not purchased the Veloce Quattro in the past, when purchasing an infringing ZeBu was an option, tells us little about what it would have done had ZeBu been pulled from the market. The jury heard several pieces of evidence that would have allowed it to determine what the Intel processor group would have done had ZeBu been pulled from the shelves. First, the jury heard testimony that at least some Intel groups had purchased Veloce Quattro emulators in the past. *See, e.g.*, Pl.'s Response [809] at Ex. C; Trial Ex. 458 at lines 866, 931, 975, and 978; Trial Tr. [737] at 691:4–6. Second, Synopsys's only argument for why the Veloce Quattro was an unacceptable alternative to ZeBu for Intel's processor group was that it lacked sufficient capacity—not speed, price, performance or foot print. However, the jury heard testimony from Dr. Degnan that it was possible to connect several emulators together to create increased capacity, and that many consumers in the emulator market were doing just that. Trial Tr. [737] at 733:18–22. Although Synopsys attacked the reliability of that testimony, it never objected to Dr. Degnan presenting it to the jury. Therefore, on the evidence the jury received, it could have reasonably concluded that even if at all times a single Veloce Quattro had insufficient capacity to meet Intel's needs, but for Synopsys's infringement Intel would have bought multiple Veloce Quattro emulators and simply connected them together to overcome their capacity shortcomings. Nothing about Mentor's use of the Veloce Quattro in its damages calculation fatally flaws the jury award.

III. Motion for New Trial on Damages

Synopsys's Motion for New Trial on Damages [790] is DENIED. For the reasons stated on the record, I reject Synopsys's arguments that: (1) Synopsys was prejudiced by a last minute change in the jury instructions; (2) the lost profits and two-supplier market instructions were clearly erroneous; and (3) the jury verdict awards Mentor double recovery. The only open issue after oral argument was whether or not *Ericsson, Inc. v. D-Link Systems, Inc.* would demonstrate that the lost profits instructions were clearly erroneous. 773 F.3d 1201 (Fed. Cir. 2014). It did not.

Synopsys argues that I should focus on the portion of *Ericsson* that merely recites the default rule that a patent owner who proves infringement is only entitled to receive damages based on the value of the patent feature—no value from non-patented features should be added. However, as both sides are well aware, there is an exception to this rule. In *State Industries, Inc. v. Mor-Flo Industries, Inc.*, the Federal Circuit held, "the entire market value rule . . . permits recovery of damages based on the value of the entire apparatus containing several features, where the patent related feature is the basis for consumer demand." 883 F.2d 1573, 1580 (Fed. Cir. 1989). The court went on to cite *Kori Corp. v. Wilco Marsh Buggies & Draglines, Inc.* for the proposition that the entire market value rule is properly applied when the nonpatented devices cannot be sold without the patented features. *Id.* (citing 761 F.2d 649, 656 (Fed. Cir.

1985) ("The ultimate determining factor is whether the patentee or its licensee can normally anticipate the sale of the unpatented components together with the patented components")). This case fits within the rule stated in *State Industries* and *Kori Corp*. There was no evidence at trial that consumers could, or ever did, purchase the unpatented features separate from the patented features. Consumers were presented with an emulator and they either bought it or they did not. There was no ability to separate the patented feature from the unpatented features and purchase some but not all of the features. Because Mentor proved at trial that it could normally anticipate the sale of the unpatented components together with the patented components, the lost profits instruction in this case did not offend or contradict the entire market value rule. The *Ericsson* recitation of the default rule does nothing to change that conclusion.

CONCLUSION

For the reasons stated above, Mentor's Motion for Accounting [783] is DENIED. Synopsys's Motion for JMOL [787] is GRANTED with respect to Mentor's contributory infringement claim and DENIED in all other respects. Synopsys's Motion for New Trial on Damages [790] is DENIED.

IT IS SO ORDERED.

DATED this __11th__ day of March, 2015.

/s/ Michael W. Mosman____
MICHAEL W. MOSMAN
United States District Judge

Permanent Injunction, Dated March 12, 2015 (DKT 844-1)

UNITED STATES DISTRICT COURT

DISTRICT OF OREGON

PORTLAND DIVISION

MENTOR GRAPHICS CORPORATION, an Oregon corporation,

Plaintiff and Counter-Defendant,

v.

EVE-USA, INC., a Delaware corporation, and **SYNOPSYS EMULATION AND VERIFICATION S.A.**, formed under the laws of France,

Defendants and Counter-Plaintiffs.

SYNOPSYS, INC., a Delaware corporation**,** **EVE-USA, INC.,** a Delaware corporation**,** and **SYNOPSYS EMULATION AND VERIFICATION S.A.,** formed under the laws of France,

Plaintiffs and Counter-Defendants,

v.

MENTOR GRAPHICS CORPORATION, an Oregon corporation,

Defendant and Counter-Plaintiff.

Civil Action No. 3:10-CV-954-MO (LEAD)
Civil Action No. 3:12-CV-1500-MO
Civil Action No. 3:13-CV-579-MO

PERMANENT INJUNCTION

I. PROHIBITED ACTIVITIES

IT IS HEREBY ORDERED that Synopsys, Inc., EVE-USA, Inc., and Synopsys Emulation and Verification, S.A. ("Synopsys") and their successors, assigns, officers, agents, servants, attorneys and employees, and persons in active concert or participation with them (including affiliated entities), who have actual notice of this injunction (the "Enjoined Parties"), are hereby permanently enjoined from infringing, inducing the infringement of, or contributing to the infringement of claims 1, 24, 26, 27, and 28 of United States Patent No. 6,240,376 (the "'376 Patent"), until the expiration of the '376 Patent, as follows:

(a) The Enjoined Parties are enjoined from making, using, selling, licensing, or leasing, or offering to sell or lease in the United States, or importing into the United States, or causing or inducing others to make, use, sell, license, or lease, or offer to sell or license or lease in the United States, or import into the United States, (i) ZeBu Server, ZeBu Server 2, ZeBu Server 3, and ZeBu Blade emulators, and variations thereof that are not more than colorably different, incorporating flexible probes or value change probes or variations thereof that are not more than colorably different, or (ii) software for ZeBu Server, ZeBu Server 2, ZeBu Server 3, and ZeBu Blade emulators, and variations thereof that are not more than colorably different, incorporating flexible probes or value change probes or variations thereof that are not more than colorably different (collectively, the "Enjoined Products"); provided, however, that for Enjoined Products sold prior to October 10, 2014, the Enjoined Parties may continue to provide service and support;

(b) Nothing in this order precludes providing service and support as to any other feature or capability of the Enjoined Products;

(c) The Enjoined Parties are enjoined from selling or leasing the Enjoined Products outside the United States for use by persons located inside the United States;

(d) The Enjoined Parties are enjoined from using from within the United States, or causing or inducing others to use from within the United States, Enjoined Products located outside the United States;

(e) The Enjoined Parties must notify their customers that Enjoined Products located outside the United States may not be imported into or used from within the United States by including the following Notice and a copy of this Permanent Injunction with the bill of sale for, or the shipment of, any Enjoined Product shipped, delivered, installed, or otherwise located outside the United States:

> **This Product is subject to a Permanent Injunction entered by the U.S. District Court for the District of Oregon (copy enclosed). Among other things, this Product may not be imported into the United States or accessed remotely by persons physically located in the United States.**

II. PERMISSIBLE "DESIGN AROUND" EFFORTS

Nothing in this Permanent Injunction prohibits the Enjoined Parties from engaging in the design, development, or testing of any product or service that does not infringe the '376 Patent. Nothing in this Permanent Injunction prohibits the Enjoined Parties from modifying any Enjoined Product to eliminate infringement, or from engaging in design, development, or testing for that purpose. Nothing in this Permanent Injunction prohibits the Enjoined Parties from making, using, selling or leasing, offering to sell or lease, or importing any Enjoined Product that has been modified to eliminate infringement.

III. NOTICE

IT IS FURTHER ORDERED that, within 15 days from the date of this signed Permanent Injunction, Synopsys shall provide a copy of this Permanent Injunction to the Enjoined Parties, including any and all manufacturers, distributors, retailers, service providers, licensees, and other persons who have been, or are reasonably expected to be, directly or indirectly involved in the making, using, selling or leasing, offering to sell or lease, or importing of any Enjoined Product.

IT IS FURTHER ORDERED that, within 15 days from the date of this signed Permanent Injunction, Synopsys shall file with the Court and serve on all parties a notice stating the names and addresses of each party that it has notified in compliance with this section.

IV. CONTINUING JURISDICTION

The Court specifically retains jurisdiction to enforce, modify, extend, or terminate this Permanent Injunction as the equities may require upon a proper showing, and to adopt procedures for resolution of any dispute over whether a product not specifically covered by this Permanent Injunction is more than colorably different from the adjudged infringing products. This injunction and the Court's continuing jurisdiction shall expire at such a time as the '376 Patent Expires.

Dated: March 12, 2015

/s/ Michael W. Mosman
Hon. Michael W. Mosman
Judge, United States District Court

U.S. Patent No. 6,132,109

US006132109A

United States Patent [19]

Gregory et al.

[11] **Patent Number: 6,132,109**

[45] **Date of Patent: *Oct. 17, 2000**

[54] **ARCHITECTURE AND METHODS FOR A HARDWARE DESCRIPTION LANGUAGE SOURCE LEVEL DEBUGGING SYSTEM**

[75] Inventors: **Brent Gregory**; **Trinanjan Chatterjee**; **Jing C. Lin**; **Srinivas Raghvendra**, all of Sunnyvale; **Emil Girczyc**, Los Altos; **Paul Estrada**, Mountain View; **Andrew Seawright**, Cupertino, all of Calif.

[73] Assignee: **Synopsys, Inc.**, Mountain View, Calif.

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **08/253,470**

[22] Filed: **Jun. 3, 1994**

Related U.S. Application Data

[63] Continuation-in-part of application No. 08/226,147, Apr. 12, 1994, abandoned.

[51] **Int. Cl.**^{7} **G06F 9/445**; G06F 9/45; G06F 15/00

[52] **U.S. Cl.** **395/704**; 364/489

[58] **Field of Search** 395/700, 701, 395/704, 705; 364/578, 488, 489, 490, 491

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,546,435	10/1985	Herbert et al.	364/300
4,667,290	5/1987	Goss et al.	395/707
4,703,435	10/1987	Darringer et al.	364/489
4,827,427	5/1989	Hyduke	364/489
4,852,173	7/1989	Bahl et al.	381/43
4,866,663	9/1989	Griffin	395/500
4,868,770	9/1989	Smith et al.	364/578
4,882,690	11/1989	Shinsha et al.	364/490

(List continued on next page.)

OTHER PUBLICATIONS

Weiss, Ray, “ASIC’s forcing logic synthesis’ hand,” Electronic Engineering Times, n516, T16 (pp. 1–2), Dec. 12, 1988.

Weiss, Ray, “Designers moving toward high–level logic representation via logic synthesis—Logic synthesis edging up design hierarchy,” Electronic Engineering Times, n526, 81, (pp. 1–10), Feb. 6, 1989.

Weiss, Ray, “Synopsys fine–tunes logic synthesis,” Electronic Engineering Times, n534, 138 (pp. 1–3), Apr. 17, 1989.

Brian Ebert et al., “SeeSaw: A Verilog Synthesis Viewer,” 2nd Annual International Verilog HDL Conference, Design Excellence for Today and Tomorrow; Santa Clara, CA, Mar. 22–24, 1993, pp. 55–60.

Lis et al., “VHDL Synthesis Using Structured Modeling,” 26th ACM/IEEE Design Automation Conference, Jun. 25, 1989, pp. 606–609.

Sougata Mukherjea, et al., “Applying Algorithm Animation Techniques for Program Tracing, Debugging, and Understanding,” Proceedings 15th International Conference on Software Engineering, May 17, 1993, Baltimore, MD, pp. 456–465.

Design Analyzer Reference (per paper #9), “Text Viewer,” v.3.1, pp. 1–15, Mar. 1994.

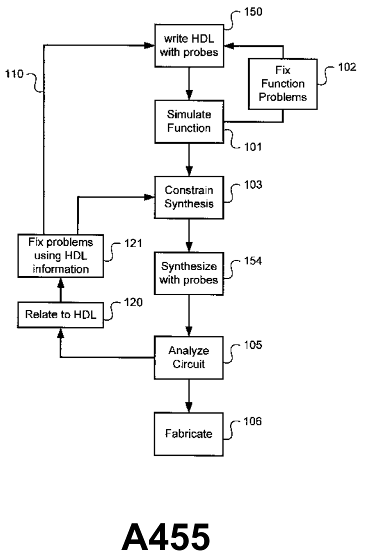
(List continued on next page.)

Primary Examiner—Tariq R. Hafiz
Assistant Examiner—Michael Pender
Attorney, Agent, or Firm—Brown Raysman Millstein Felder & Steiner LLP; Jonathan T. Kaplan

[57] **ABSTRACT**

This invention provides a method for displaying circuit analysis results corresponding to parts of the circuit near the portion of the hardware description language (HDL) specification that generated that part of the circuit. The invention also includes a method for using probe statements in the HDL specification to mark additional points in the initial circuit that should not be eliminated during optimization. This improves the ability to display circuit analysis results near the appropriate part of the HDL specification.

2 Claims, 33 Drawing Sheets



U.S. PATENT DOCUMENTS

4,907,180	3/1990	Smith	364/578
4,942,536	7/1990	Watanabe et al.	364/490
4,942,615	7/1990	Hirose	364/578
4,967,386	10/1990	Maeda et al.	364/578
4,970,664	11/1990	Kaiser et al.	364/521
5,111,413	5/1992	Lazensky et al.	364/578
5,146,583	9/1992	Matsunaka et al.	395/500
5,191,541	3/1993	Landman et al.	364/489
5,191,646	3/1993	Naito et al.	395/161
5,265,254	11/1993	Blasciak et al.	395/700
5,282,146	1/1994	Aihara et al.	364/489
5,282,148	1/1994	Poirot et al.	364/491
5,329,471	7/1994	Swoboda et al.	364/578
5,335,191	8/1994	Kundert et al.	364/578
5,377,997	1/1995	Wilden et al.	273/434
5,437,037	7/1995	Furuichi	395/700
5,446,900	8/1995	Kimelman	395/700
5,452,239	9/1995	Dai et al.	364/578
5,493,507	2/1996	Shinde et al.	395/500.35
5,530,841	6/1996	Gregory et al.	395/500
5,541,849	7/1996	Rostoker et al.	364/489
5,544,066	8/1996	Rostoker et al.	364/489
5,544,067	8/1996	Rostoker et al.	364/489
5,544,068	8/1996	Takimoto et al.	364/489
5,553,002	9/1996	Dangelo et al.	364/489
5,555,201	9/1996	Dangelo et al.	364/489
5,557,531	9/1996	Rostoker et al.	364/489
5,581,738	12/1996	Dambrowski	395/500.17

OTHER PUBLICATIONS

D. E. Thomas et al., “Algorithmic and Register–Transfer Level Synthesis: The System Architect’s Workbench,” pp. 257–274, publication date unknown.

Pure Software Inc., Pure Coverage Data Sheet (Web Page), copyright 1996.

Pure Software Inc., Quantify Data Sheet (Web Page), copyright 1996.

Pure Software Inc., Purify, Finding Run–Time Memory Errors (Web Page), coyright 1996.

Reed Hastings et al., Pure Software, Inc., Purify, Usenix White Paper on Purify (Web Page), copyright 1996.

Printout of Web Page for Centerline, Code Center, Release 4, Nov. 1994.

HDC Computer Corporation, FirstApps User’s Guide, pp. 112–116, copyright 1992.

Louis Trevillyan, “An Overview of Logic Synthesis Systems,” 1987, pp. 166–172.

Timothy Kam, “Comparing Layouts with HDL Models: A Formal Verification Technique,” 1992, pp. 588–591.

“A Programming the User Interface”, Manual of Symbolics, Inc., 4 New England Tech Center, 555 Virginia Road, Concord, MA 01742, title page, pp. iii–v and pp. 1–134, Sep. 1986.

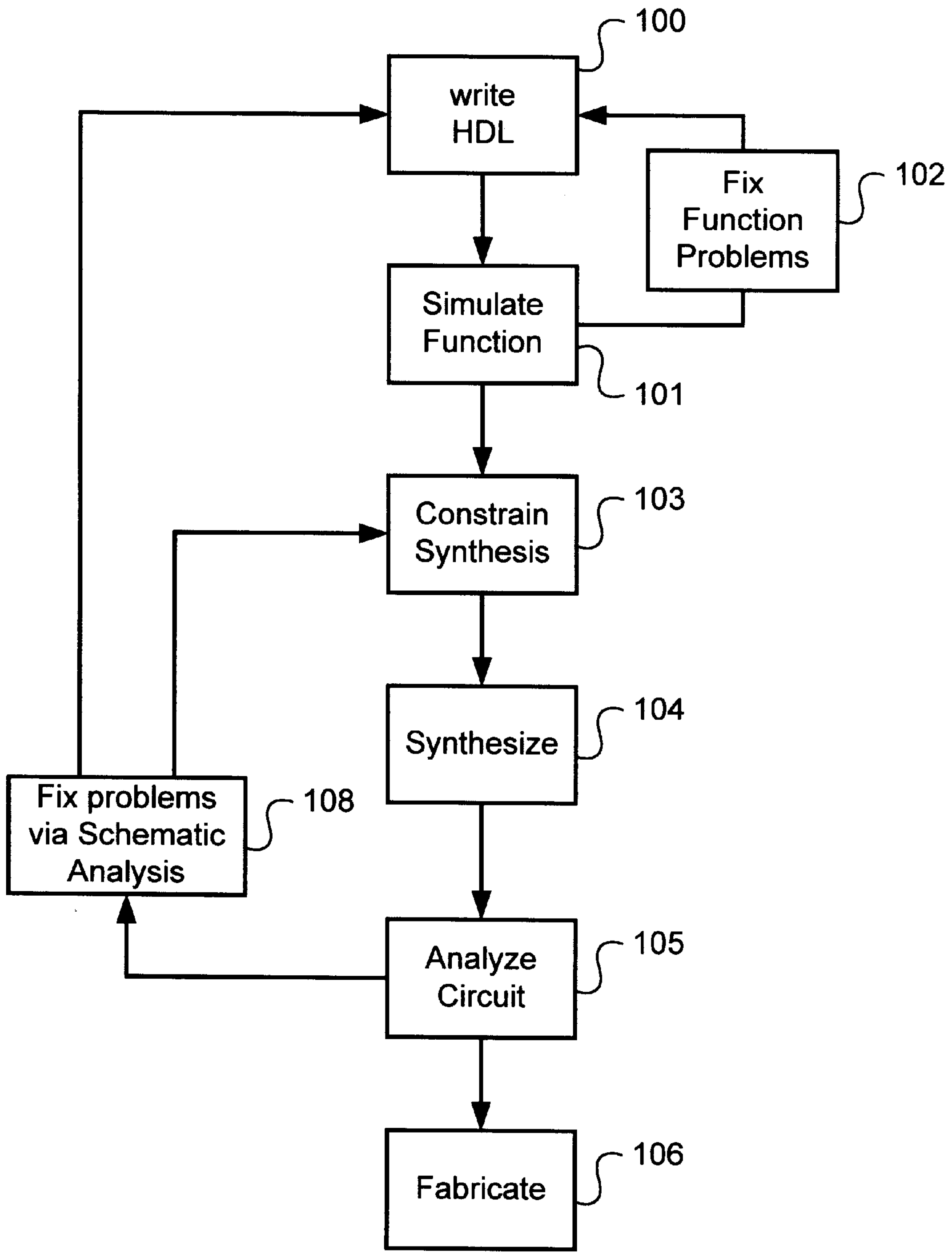

100
write HDL
102
Fix Function Problems
Simulate Function
101
Constrain Synthesis
103
Synthesize
104
Fix problems via Schematic Analysis
108
Analyze Circuit
105
Fabricate
106

Figure 1

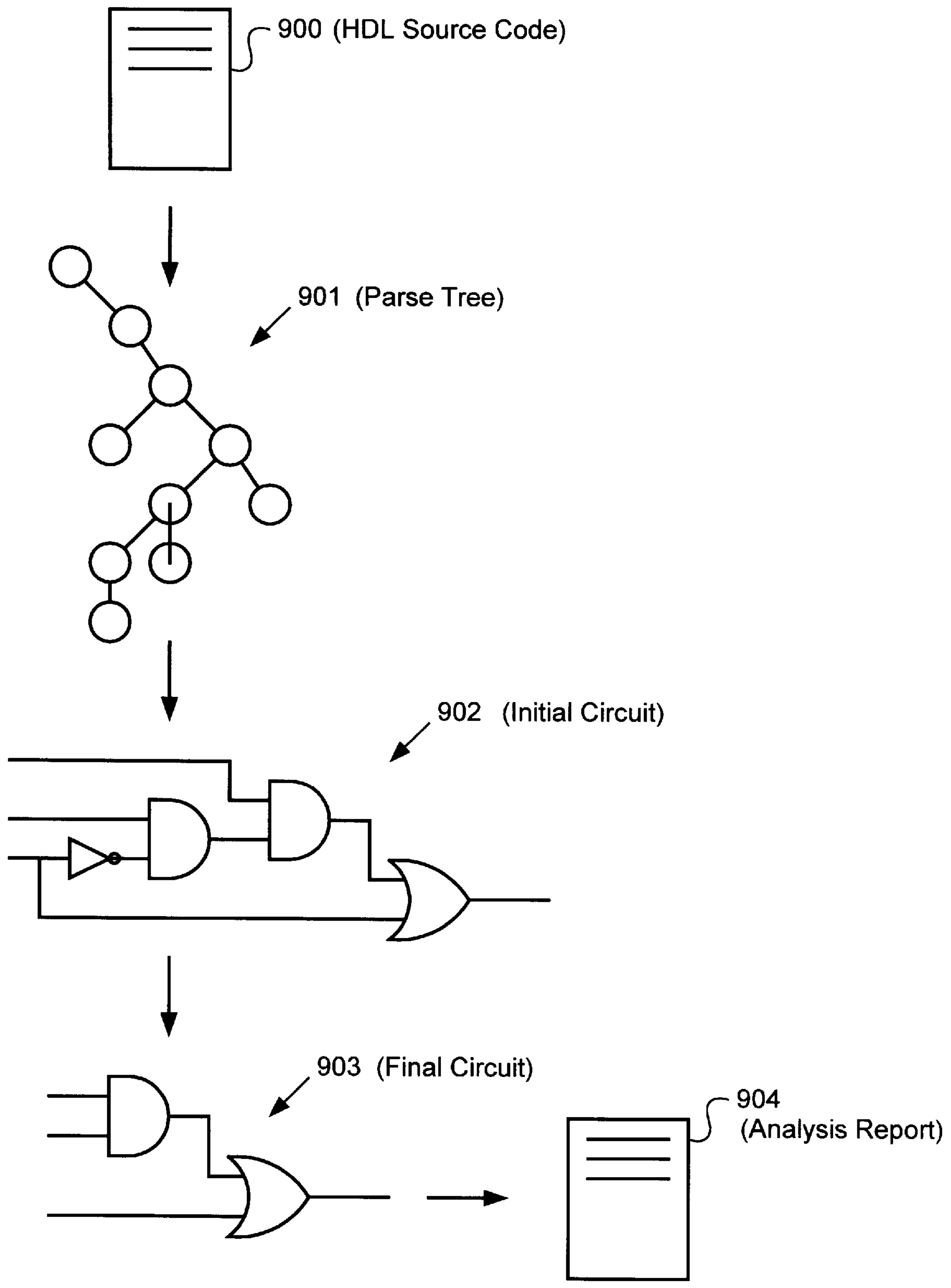

900 (HDL Source Code)
901 (Parse Tree)
902 (Initial Circuit)
903 (Final Circuit)
904 (Analysis Report)

Figure 2

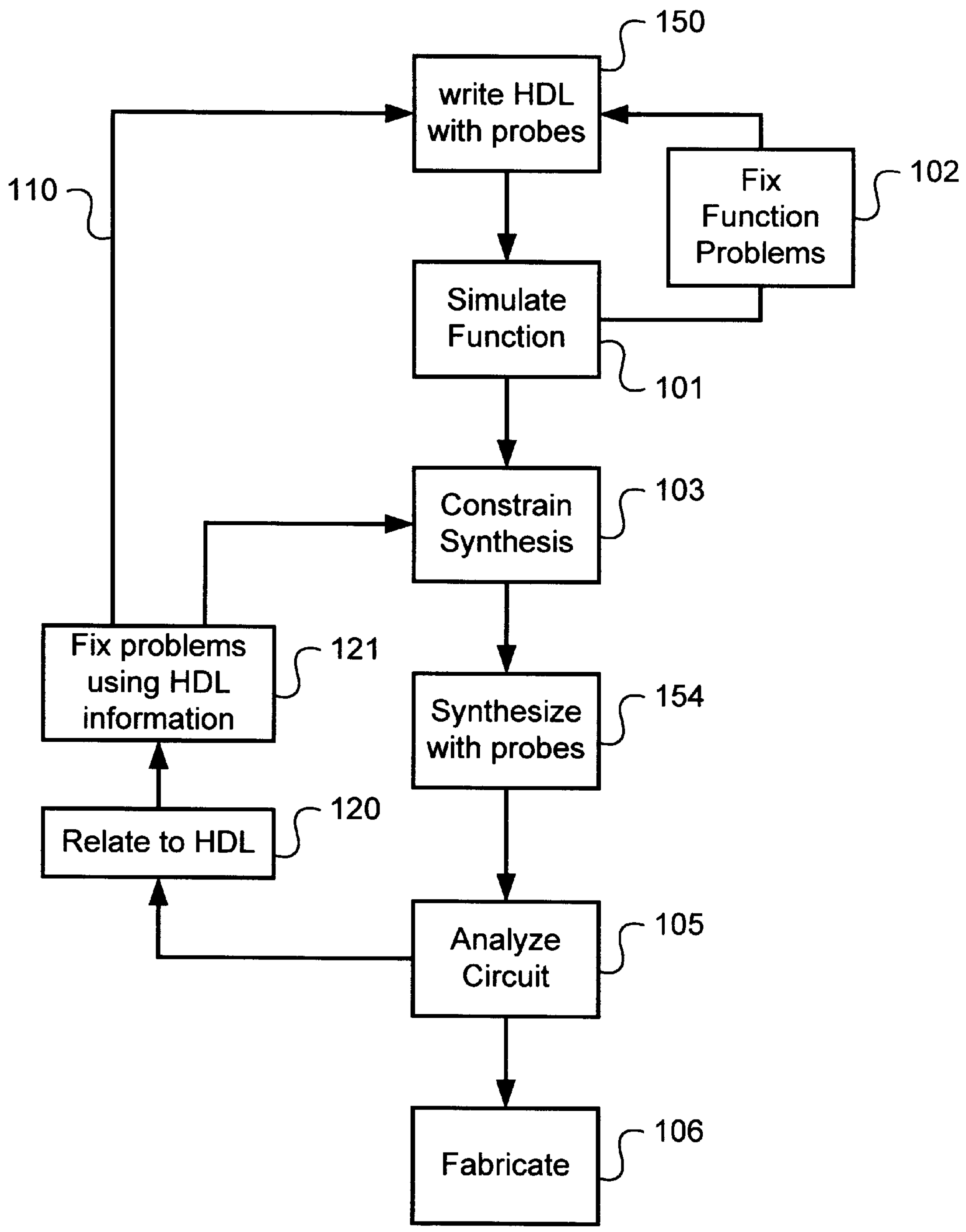
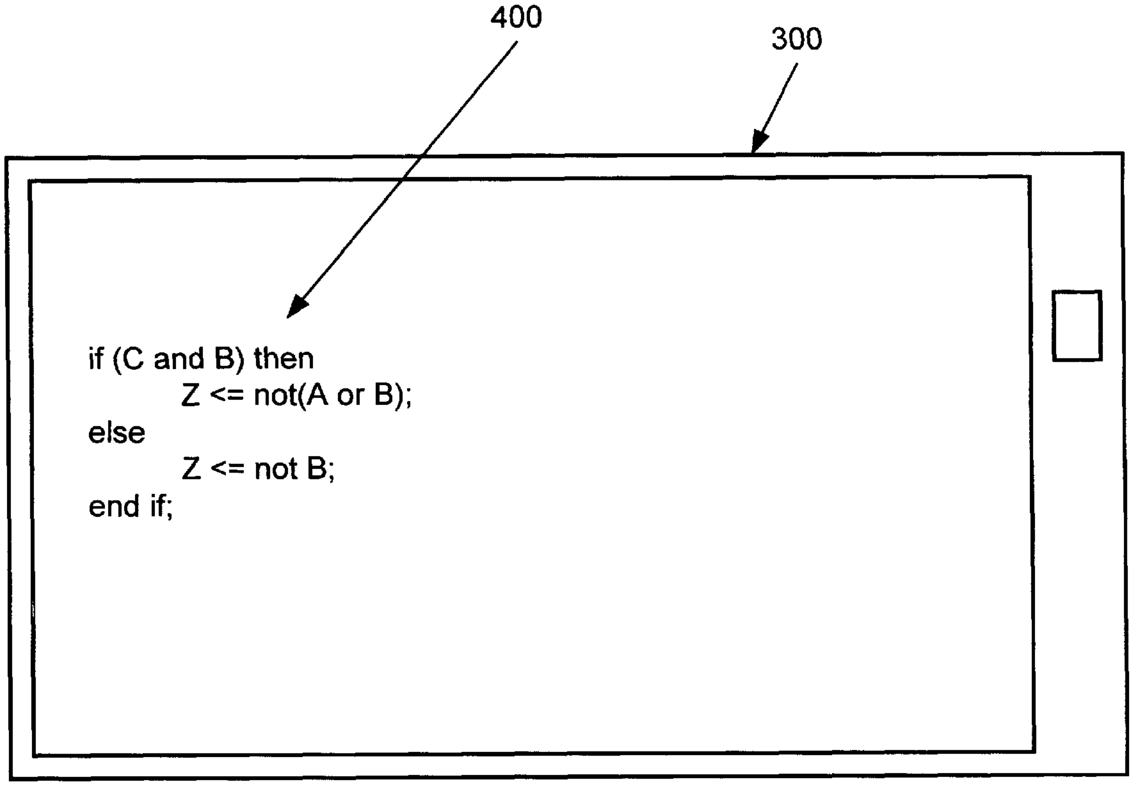

150
write HDL with probes
110
Fix Function Problems
102
Simulate Function
101
Constrain Synthesis
103
Fix problems using HDL information
121
Synthesize with probes
154
Relate to HDL
120
Analyze Circuit
105
Fabricate
106

Figure 3



```
if (C and B) then
        Z <= not(A or B);
else
        Z <= not B;
end if;
```

Figure 4

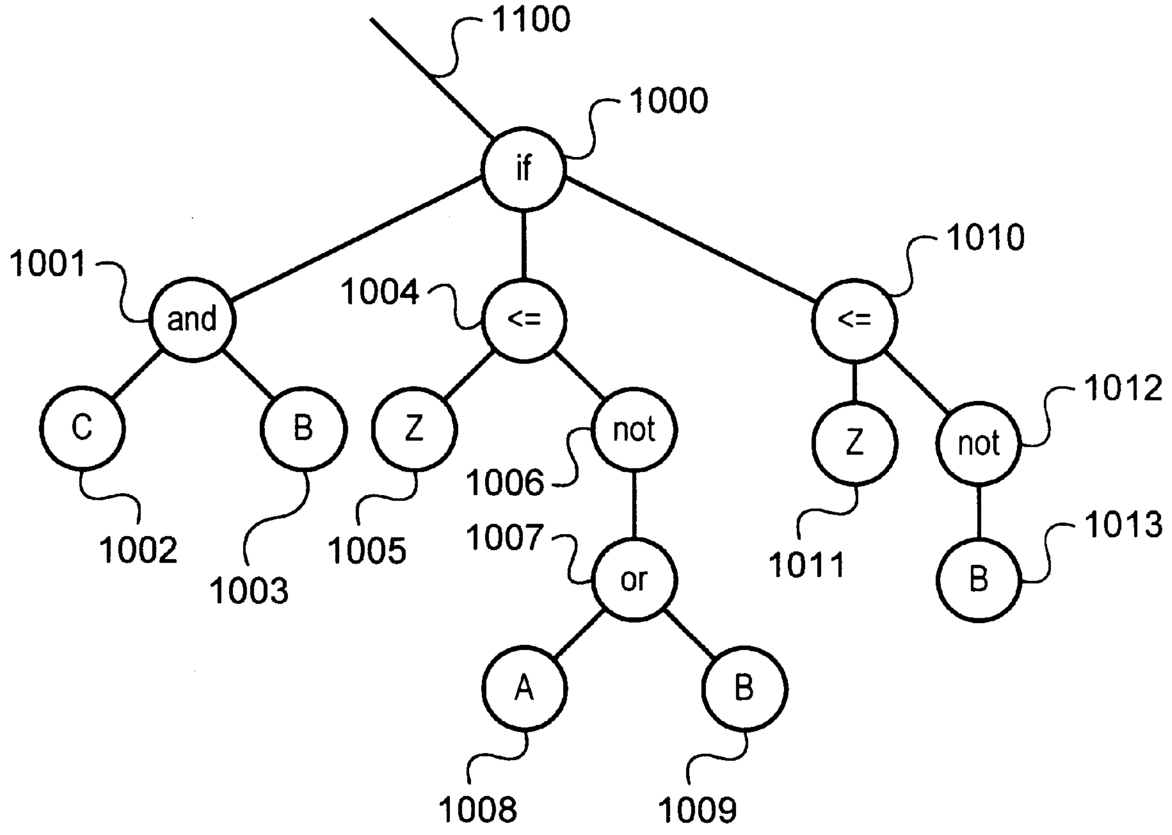

1100
1000
if
1001
and
1004
<=
1010
<=
C
B
Z
not
1006
Z
not
1012
1002
1003
1005
1007
or
1011
B
1013
A
B
1008
1009

Figure 5

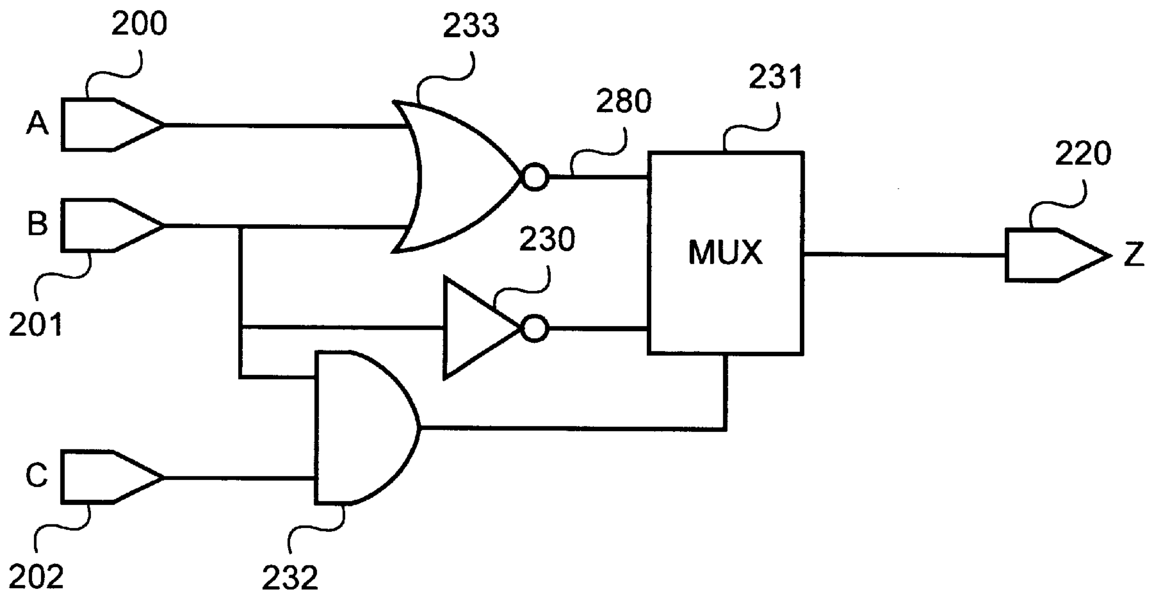


Figure 6

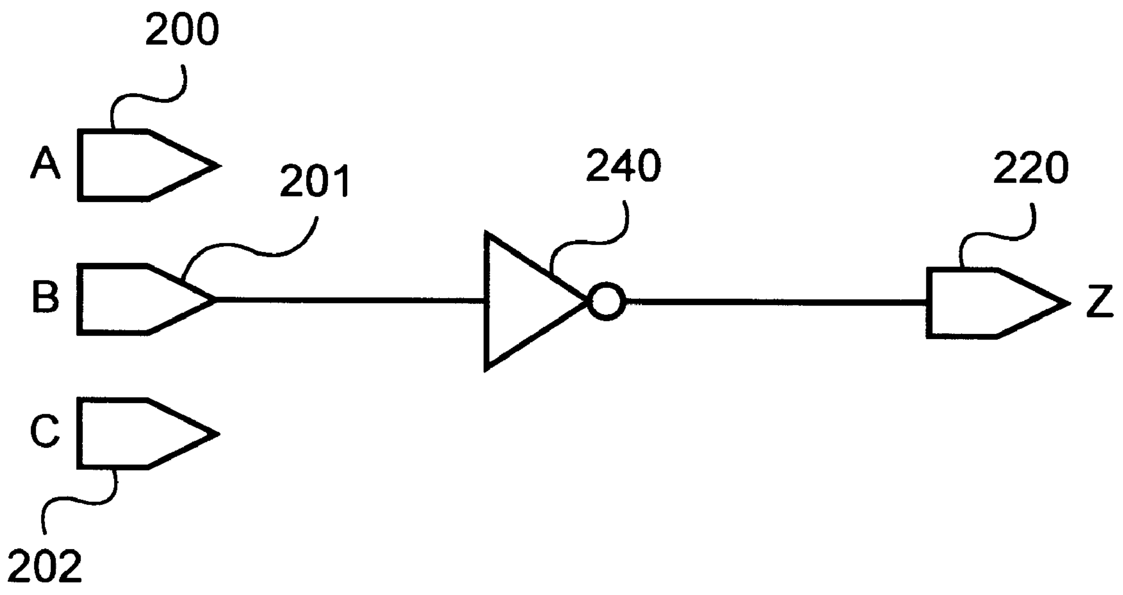
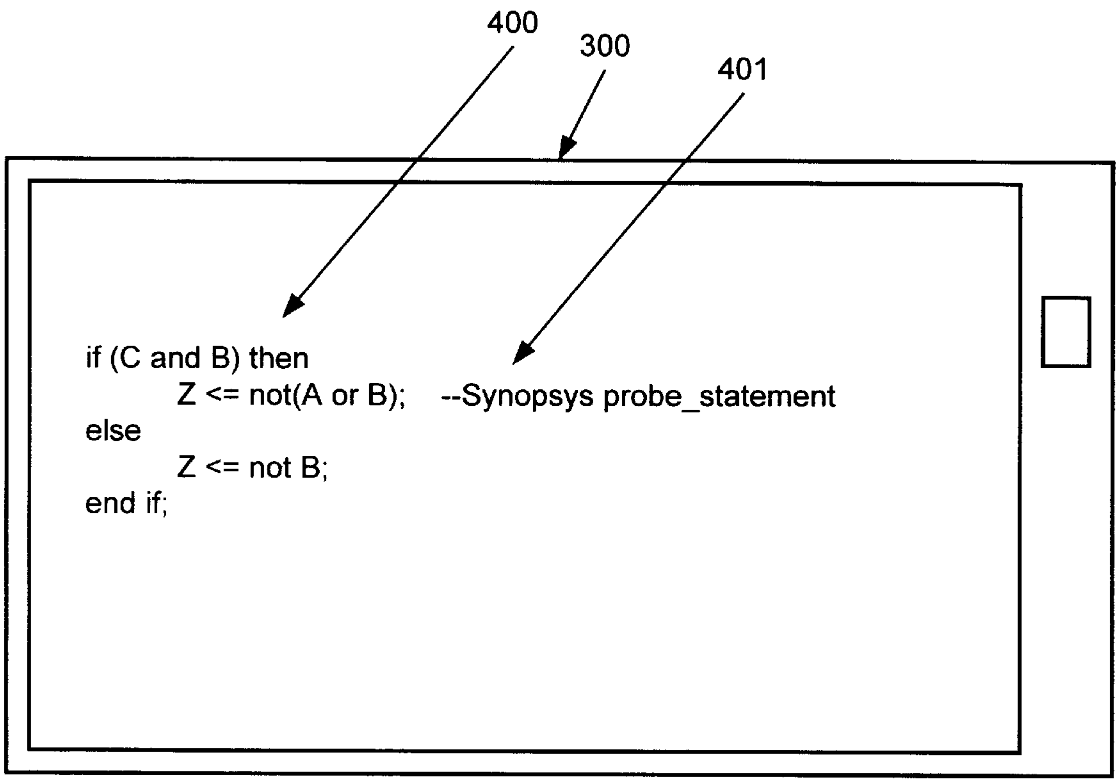
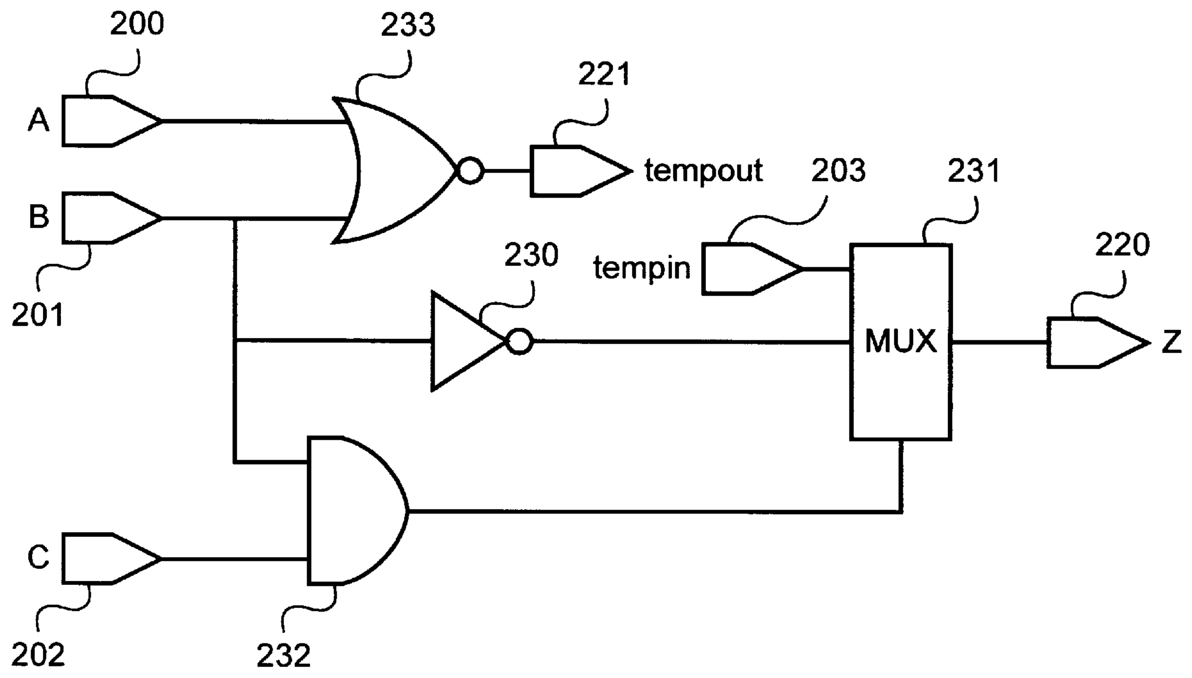


Figure 7



```
if (C and B) then
        Z <= not(A or B);    --Synopsys probe_statement
else
        Z <= not B;
end if;
```

Figure 8



200
233
221
A
tempout
203
231
B
tempin
230
220
201
MUX
Z
C
202
232

Figure 9

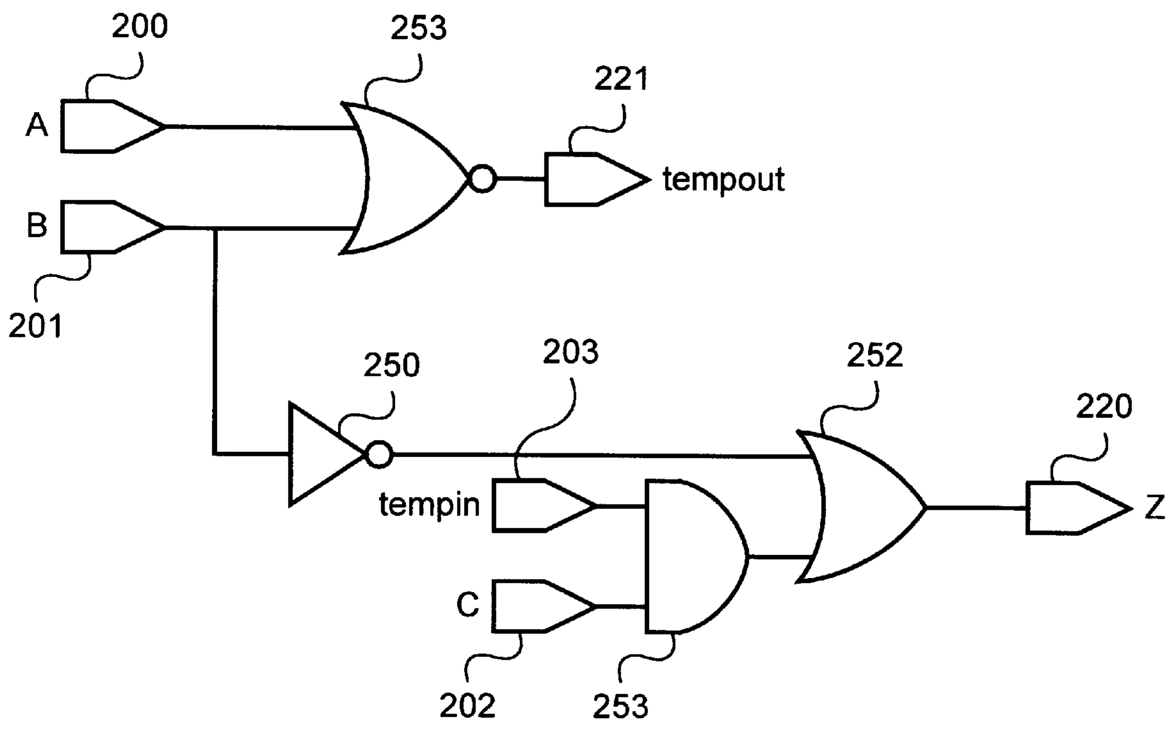
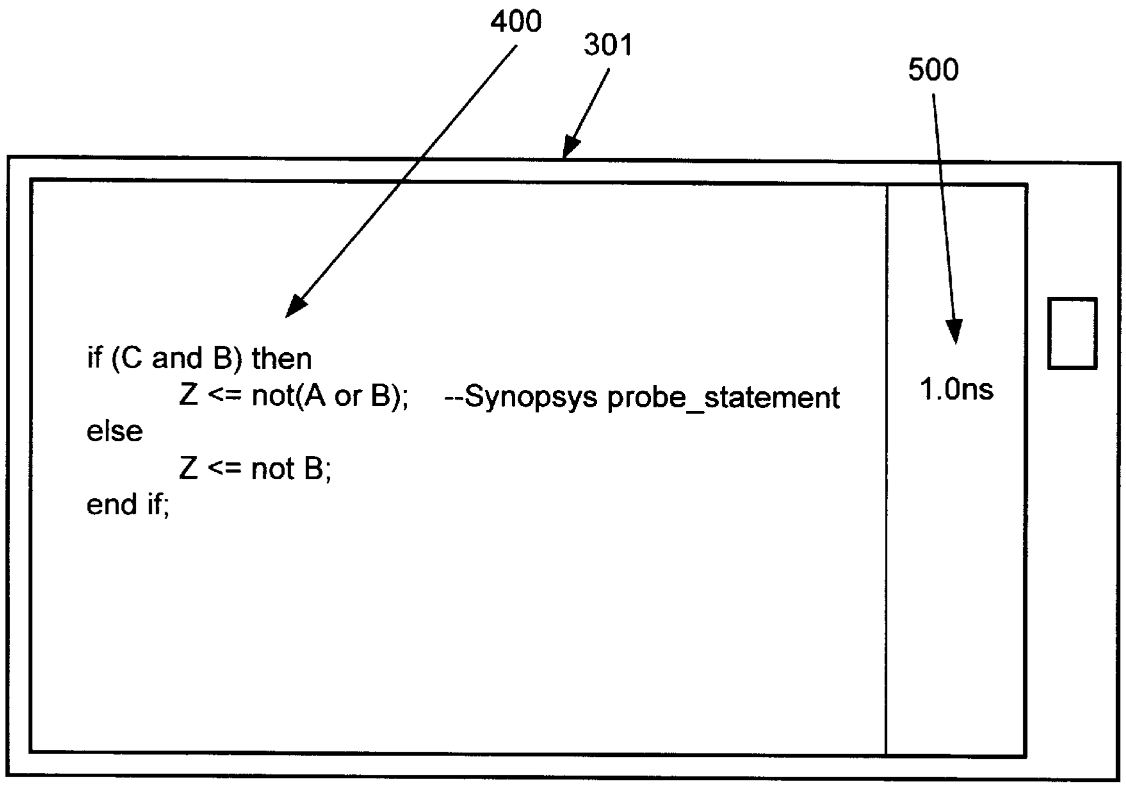
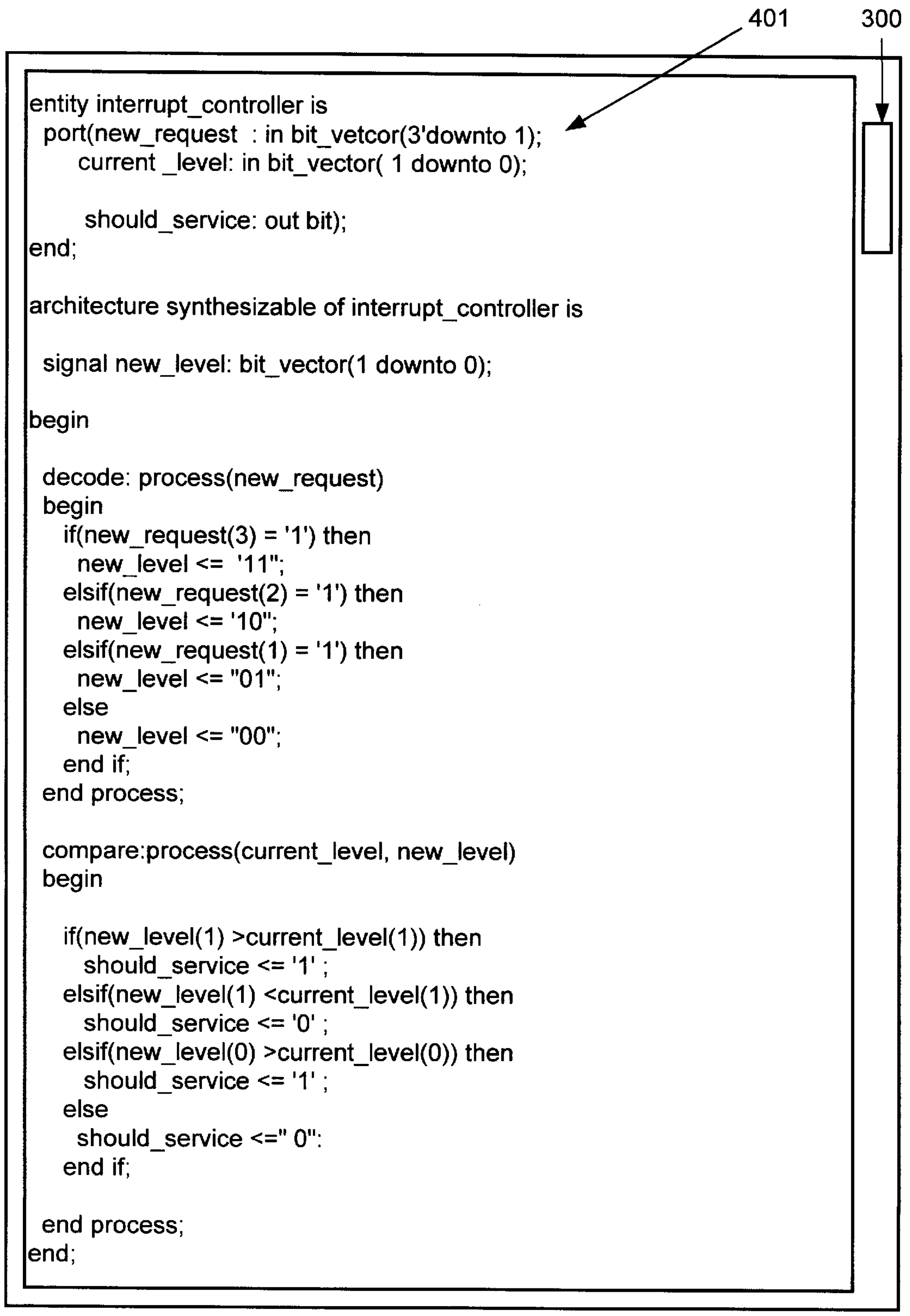


Figure 10



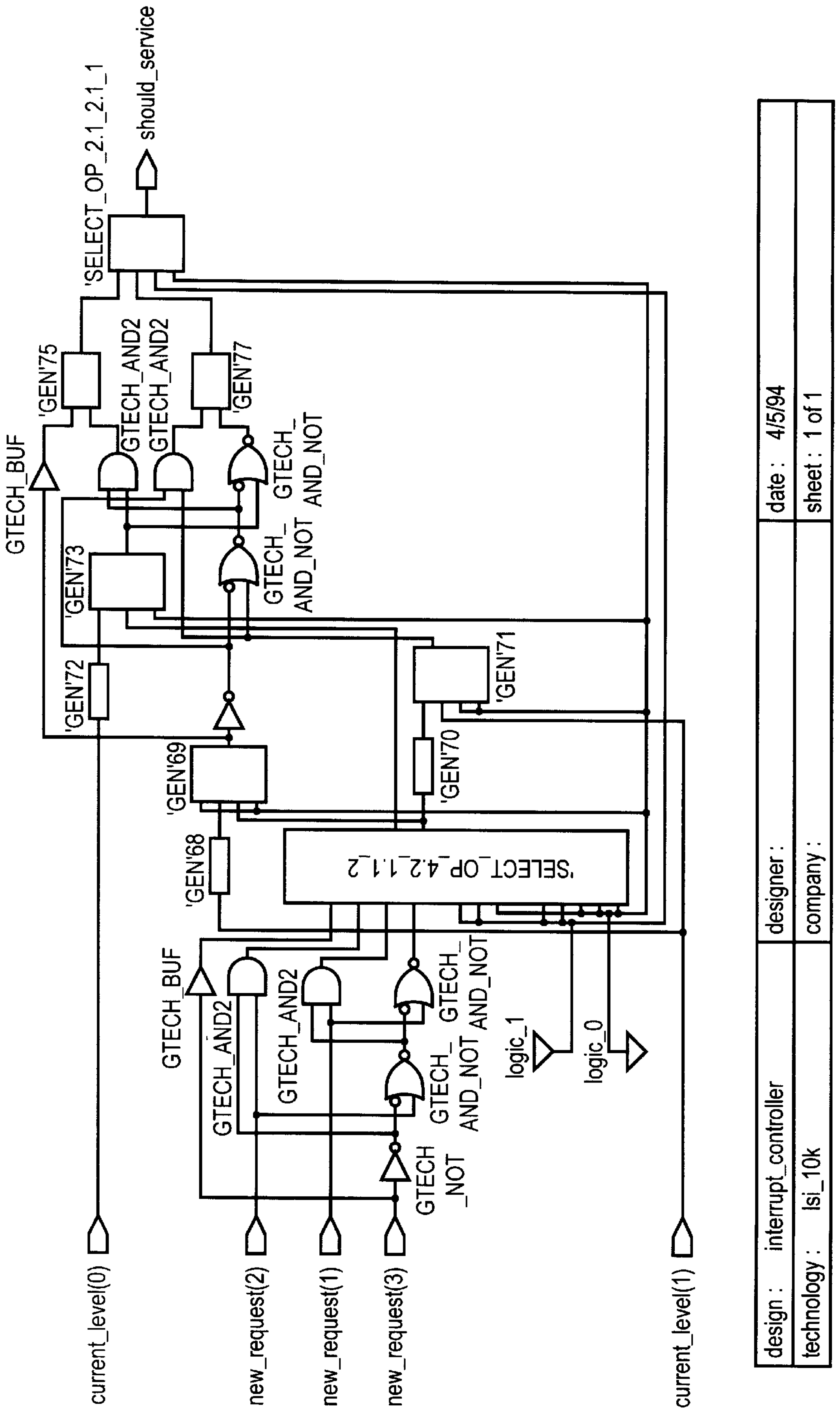
```
if (C and B) then
        Z <= not(A or B);    --Synopsys probe_statement
else
        Z <= not B;
end if;
```

Figure 11



```
entity interrupt_controller is
 port(new_request  : in bit_vetcor(3'downto 1);
     current _level: in bit_vector( 1 downto 0);

      should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

 signal new_level: bit_vector(1 downto 0);

begin

 decode: process(new_request)
 begin
   if(new_request(3) = '1') then
     new_level <=  '11";
   elsif(new_request(2) = '1') then
     new_level <= '10";
   elsif(new_request(1) = '1') then
     new_level <= "01";
   else
     new_level <= "00";
   end if;
 end process;

 compare:process(current_level, new_level)
 begin

   if(new_level(1) >current_level(1)) then
      should_service <= '1' ;
   elsif(new_level(1) <current_level(1)) then
      should_service <= '0' ;
   elsif(new_level(0) >current_level(0)) then
      should_service <= '1' ;
   else
     should_service <=" 0":
   end if;

 end process;
end;
```

Figure 12



should_service
'SELECT_OP_2.1_2.1_1
'GEN'75
'GEN'77
GTECH_AND2
GTECH_AND2
GTECH_BUF
GTECH_ AND_NOT
GTECH_ AND_NOT
'GEN'73
'GEN'72
'GEN'71
'GEN'70
'GEN'69
'GEN'68
'SELECT_OP_4.2_1.1_2
GTECH_BUF
GTECH_AND2
GTECH_AND2
GTECH_ AND_NOT
GTECH_ AND_NOT
GTECH _NOT
logic_1
logic_0
current_level(0)
new_request(2)
new_request(1)
new_request(3)
current_level(1)
design : interrupt_controller
technology : lsi_10k
designer :
company :
date : 4/5/94
sheet : 1 of 1

Figure 13

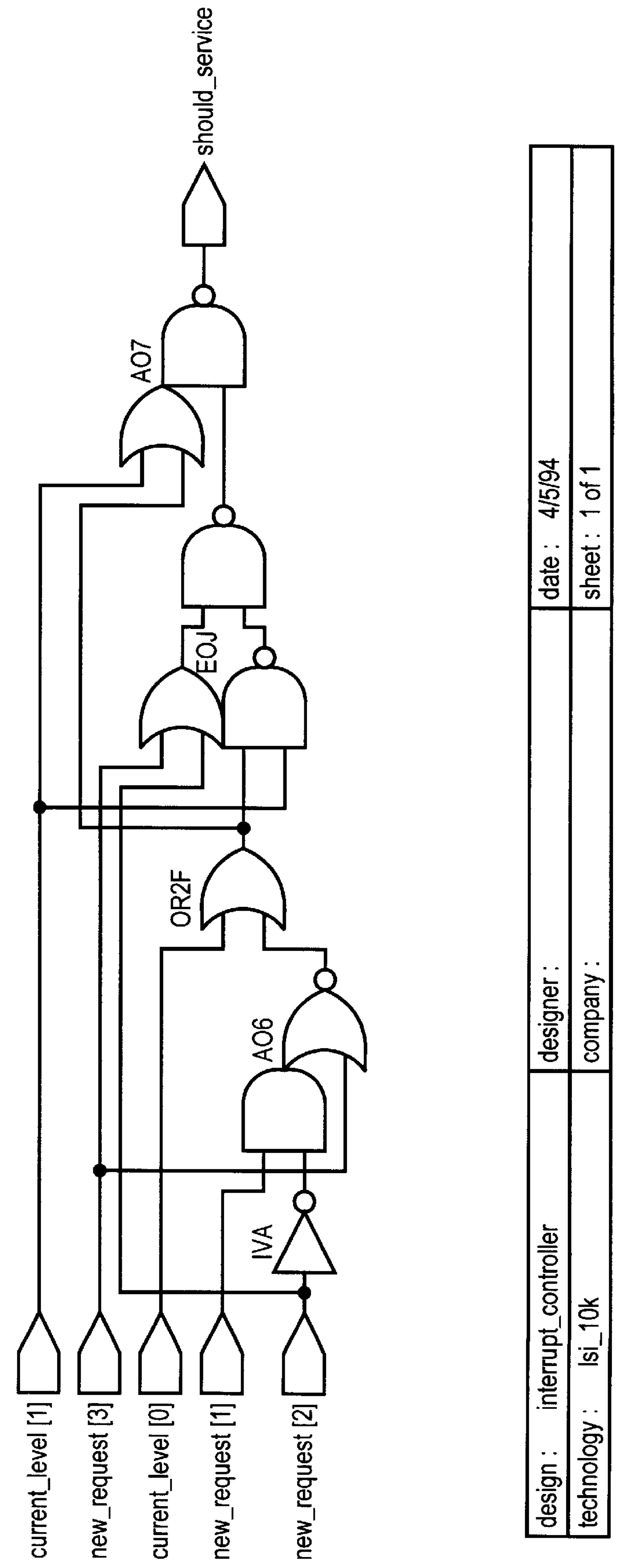
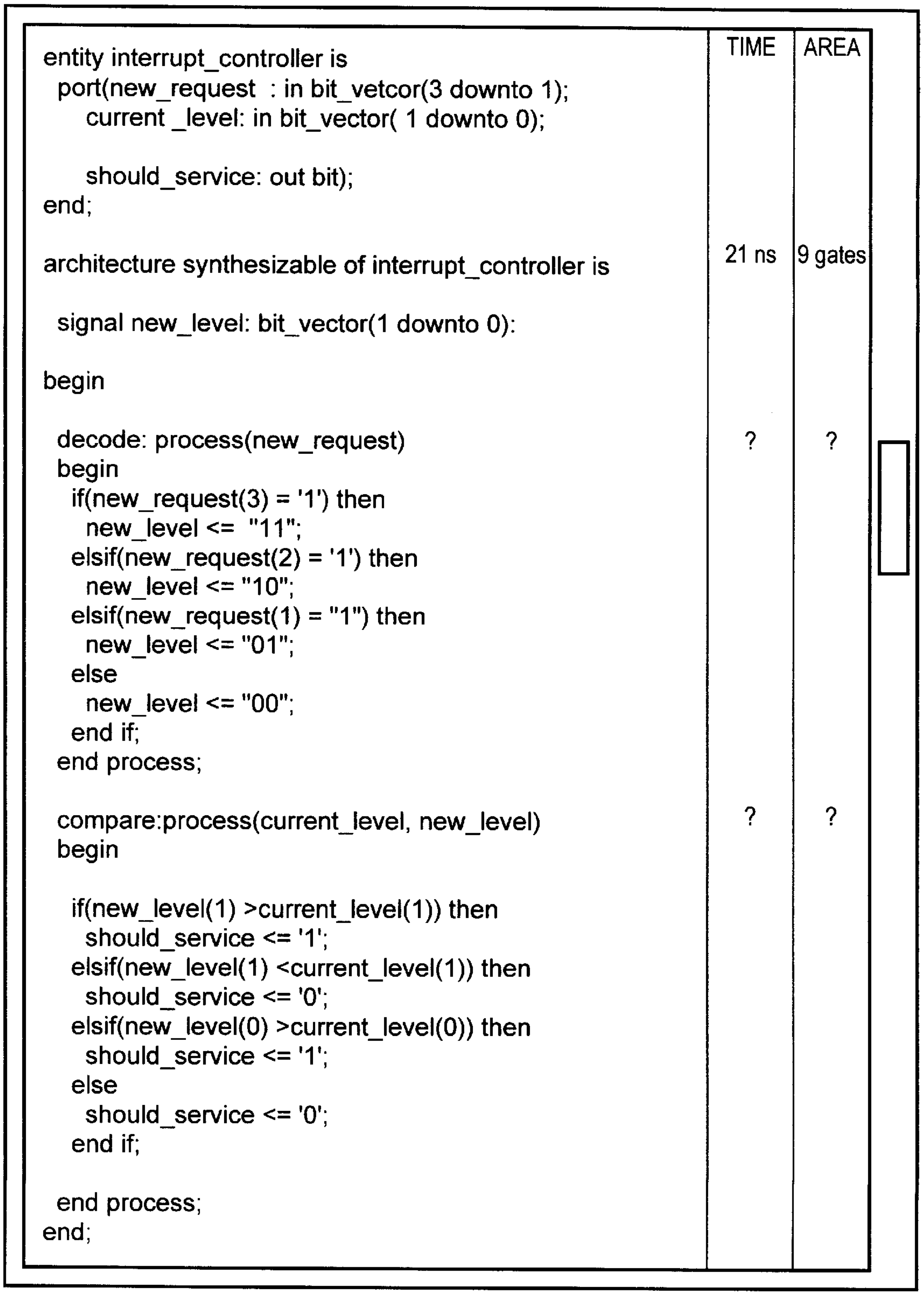
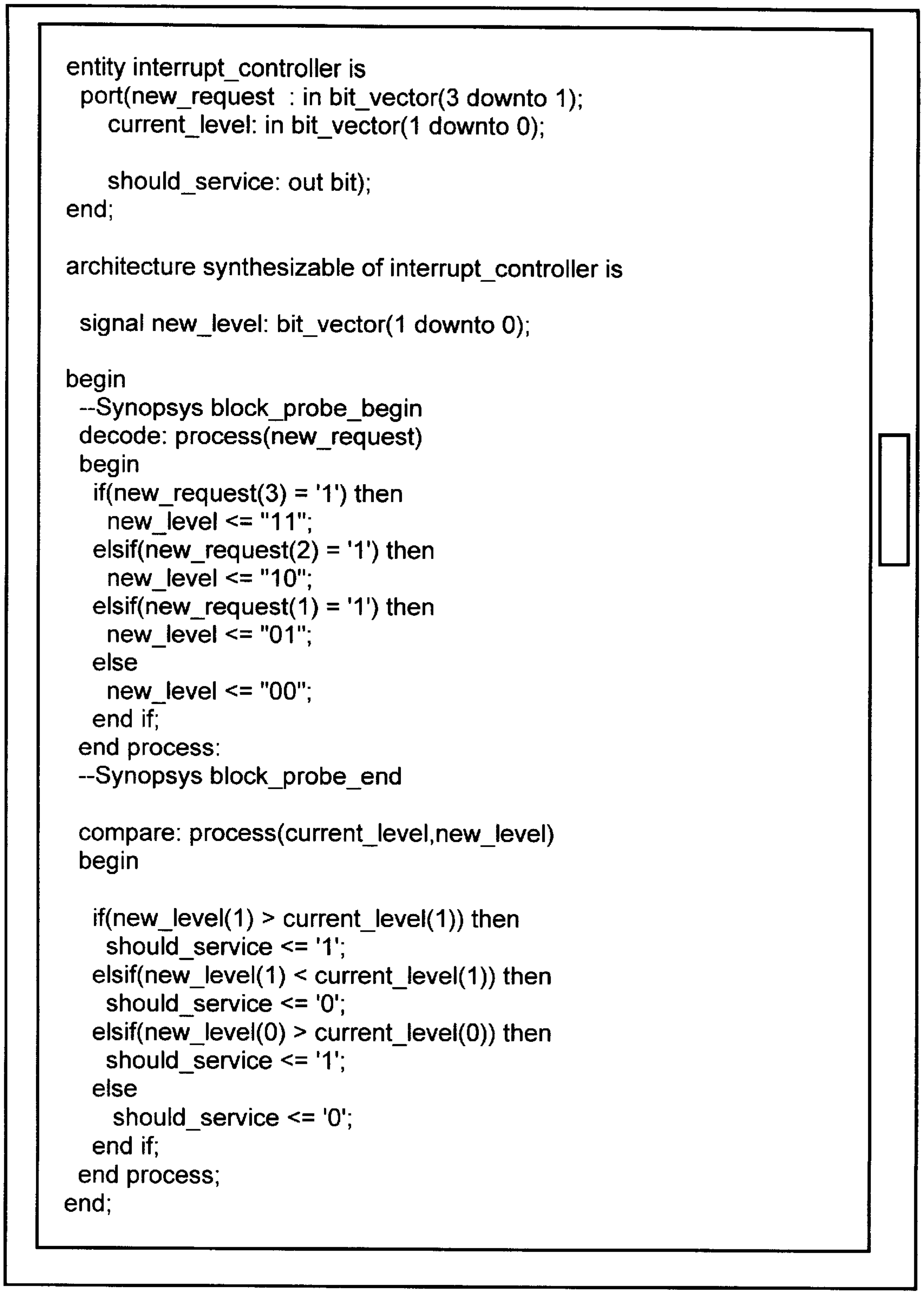

should_service
AO7
EOJ
OR2F
AO6
IVA
current_level [1]
new_request [3]
current_level [0]
new_request [1]
new_request [2]
design : interrupt_controller
technology : lsi_10k
designer :
company :
date : 4/5/94
sheet : 1 of 1

Figure 14



```
                                                                TIME    AREA
entity interrupt_controller is
  port(new_request  : in bit_vetcor(3 downto 1);
      current _level: in bit_vector( 1 downto 0);

      should_service: out bit);
end;

architecture synthesizable of interrupt_controller is          21 ns   9 gates

  signal new_level: bit_vector(1 downto 0):

begin

  decode: process(new_request)                                    ?       ?
  begin
    if(new_request(3) = '1') then
      new_level <=  "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = "1") then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process;

  compare:process(current_level, new_level)                       ?       ?
  begin

    if(new_level(1) >current_level(1)) then
      should_service <= '1';
    elsif(new_level(1) <current_level(1)) then
      should_service <= '0';
    elsif(new_level(0) >current_level(0)) then
      should_service <= '1';
    else
      should_service <= '0';
    end if;

  end process;
end;
```

Figure 15



```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
      current_level: in bit_vector(1 downto 0);

      should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin
  --Synopsys block_probe_begin
  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process:
  --Synopsys block_probe_end

  compare: process(current_level,new_level)
  begin

    if(new_level(1) > current_level(1)) then
      should_service <= '1';
    elsif(new_level(1) < current_level(1)) then
      should_service <= '0';
    elsif(new_level(0) > current_level(0)) then
      should_service <= '1';
    else
      should_service <= '0';
    end if;
  end process;
end;
```

Figure 16

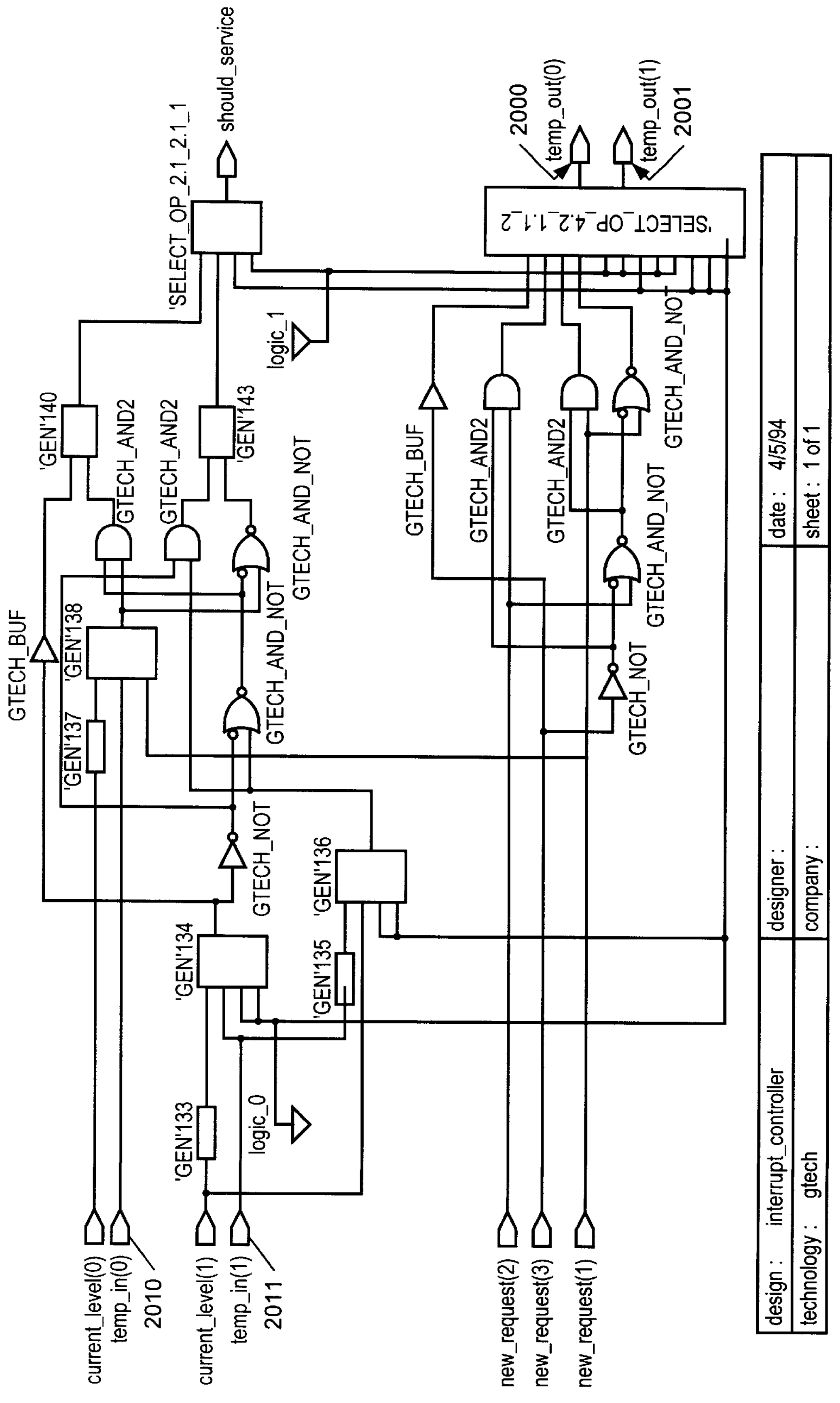

should_service
'SELECT_OP_2.1_2.1_1
2000
temp_out(0)
temp_out(1)
2001
'SELECT_OP_4.2_1.1_2
logic_1
'GEN'140
'GEN'143
GTECH_AND2
GTECH_AND2
GTECH_AND_NOT
GTECH_AND_NOT
GTECH_BUF
'GEN'138
'GEN'137
GTECH_AND_NOT
GTECH_NOT
'GEN'136
'GEN'134
'GEN'135
'GEN'133
logic_0
GTECH_BUF
GTECH_AND2
GTECH_AND2
GTECH_AND_NOT
GTECH_AND_NOT
GTECH_NOT
current_level(0)
temp_in(0)
2010
current_level(1)
temp_in(1)
2011
new_request(2)
new_request(3)
new_request(1)
design : interrupt_controller
technology : gtech
designer :
company :
date : 4/5/94
sheet : 1 of 1

Figure 17

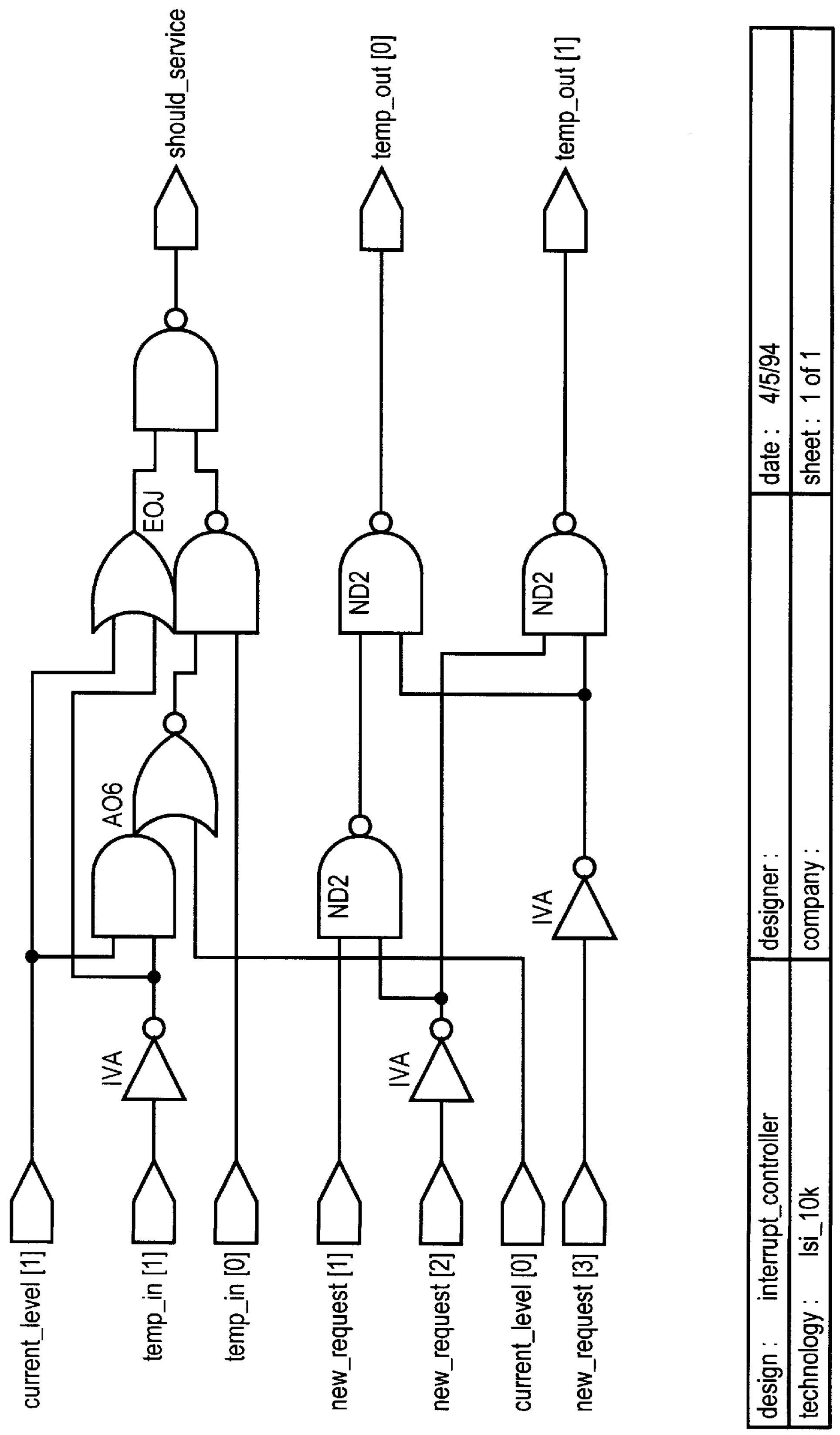
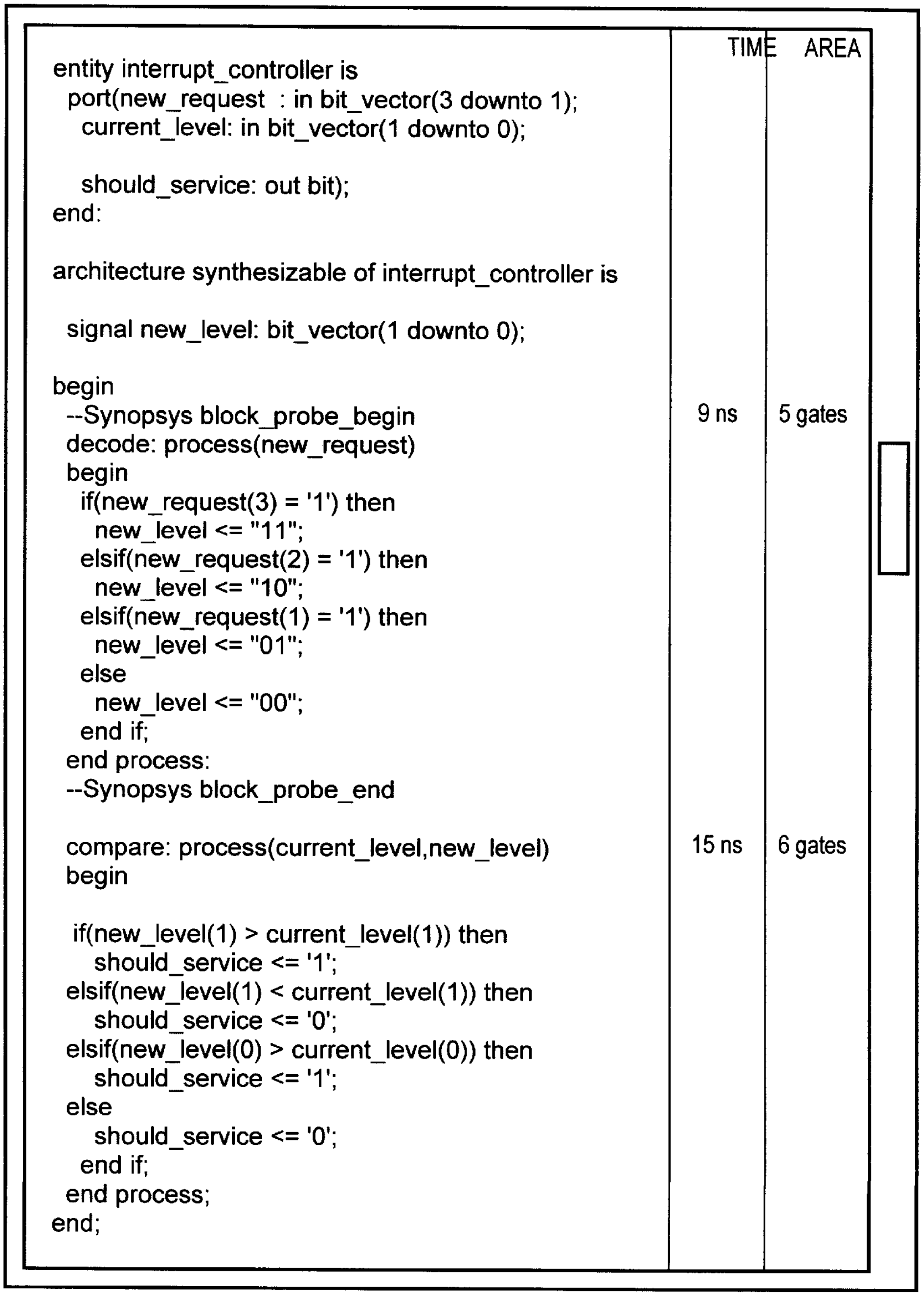

current_level [1]
temp_in [1]
temp_in [0]
new_request [1]
new_request [2]
current_level [0]
new_request [3]
IVA
IVA
IVA
AO6
EOJ
ND2
ND2
ND2
should_service
temp_out [0]
temp_out [1]
design : interrupt_controller
technology : lsi_10k
designer :
company :
date : 4/5/94
sheet : 1 of 1

Figure 18



```
                                                            TIME     AREA
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
    current_level: in bit_vector(1 downto 0);

    should_service: out bit);
end:

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin
  --Synopsys block_probe_begin                                9 ns     5 gates
  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process:
  --Synopsys block_probe_end

  compare: process(current_level,new_level)                   15 ns    6 gates
  begin

   if(new_level(1) > current_level(1)) then
      should_service <= '1';
  elsif(new_level(1) < current_level(1)) then
      should_service <= '0';
  elsif(new_level(0) > current_level(0)) then
      should_service <= '1';
  else
      should_service <= '0';
    end if;
  end process;
end;
```

Figure 19

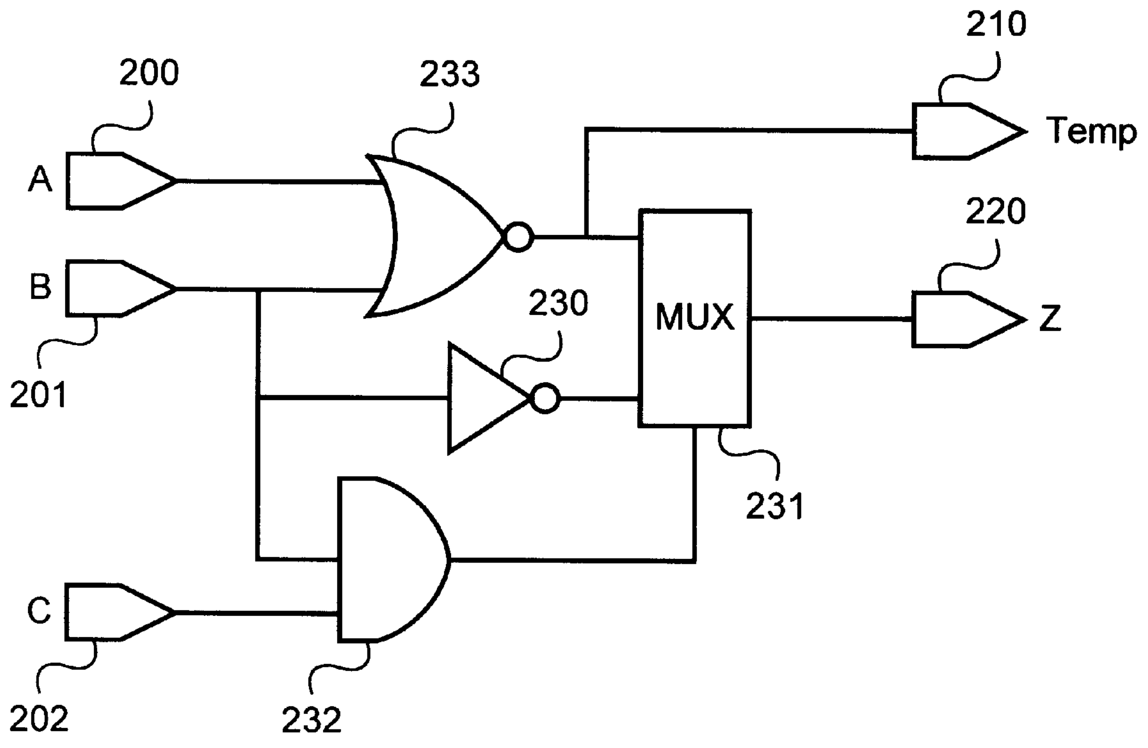


Figure 20

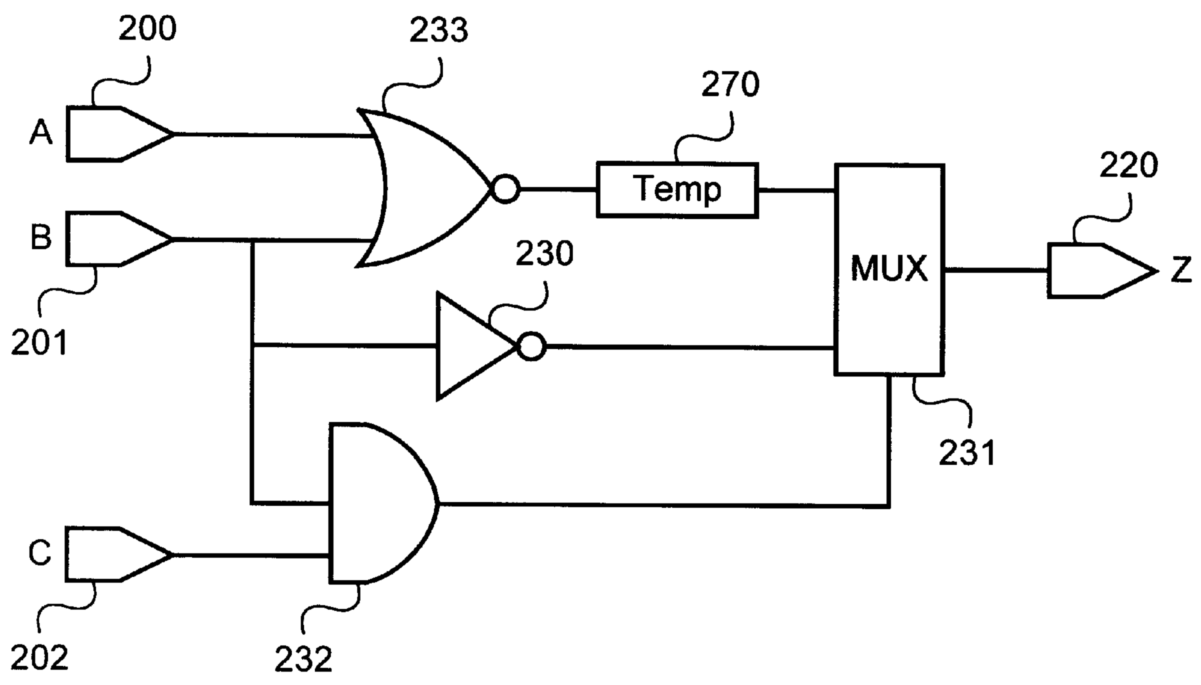
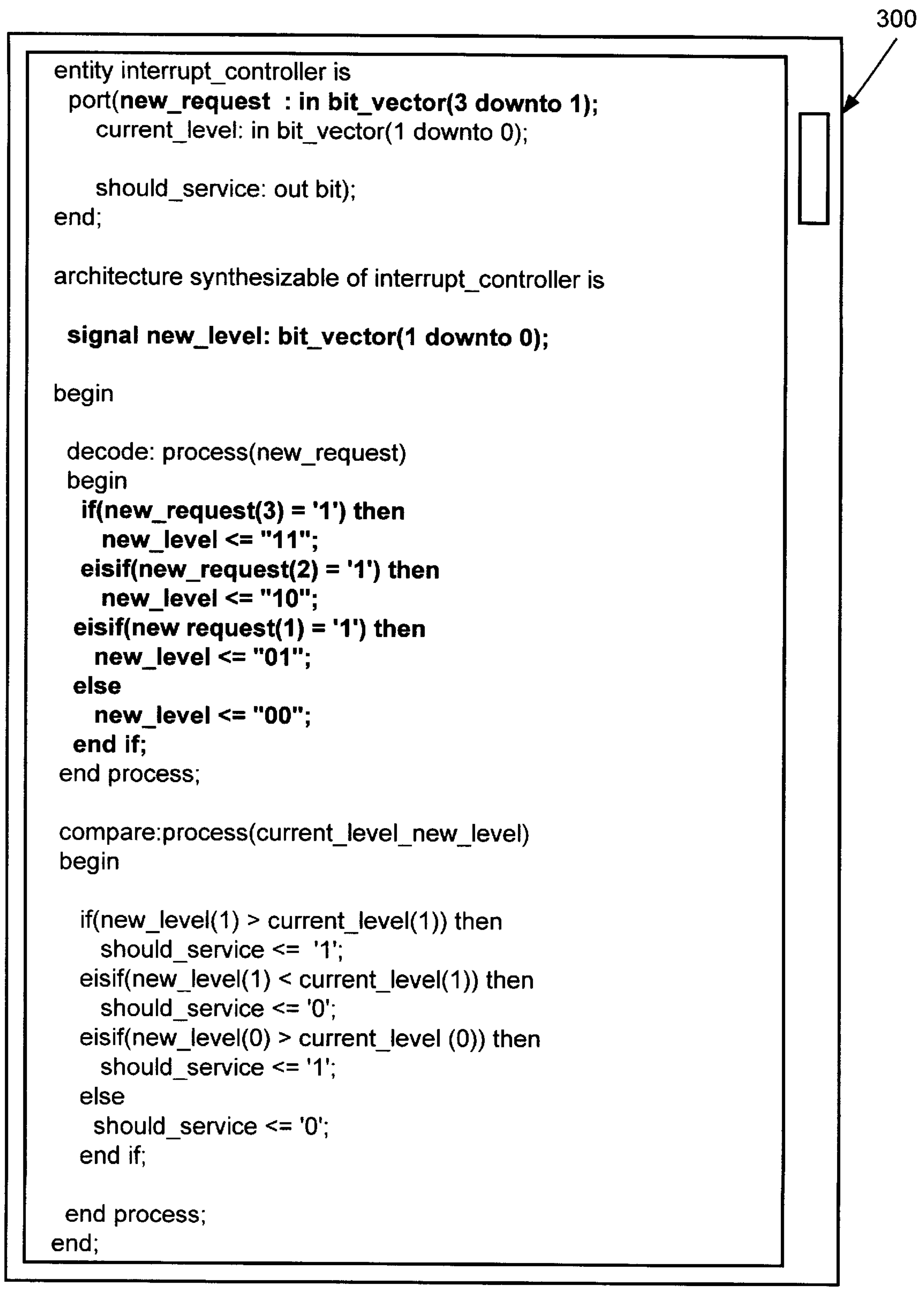
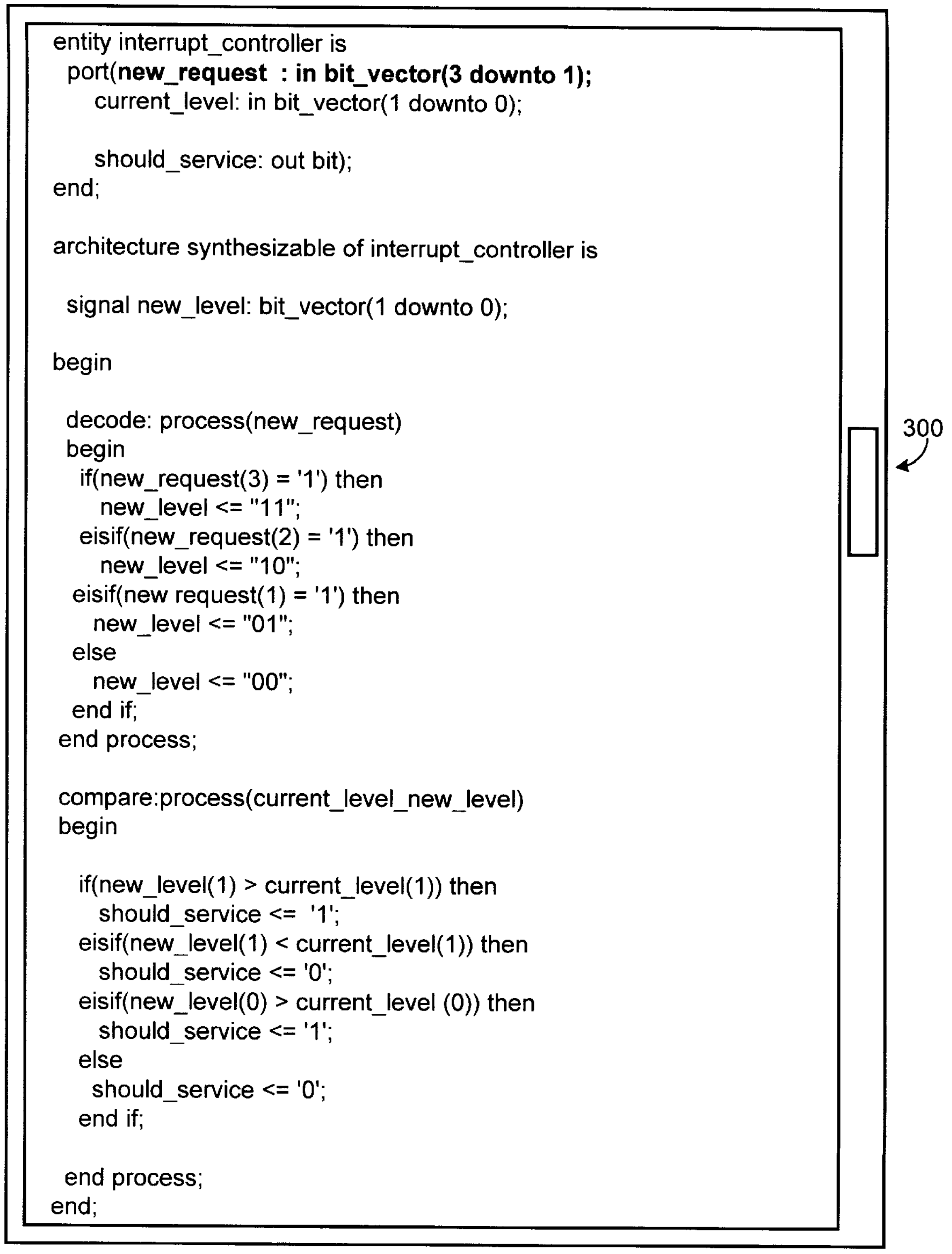


Figure 21



```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
      current_level: in bit_vector(1 downto 0);

      should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin

  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    eisif(new_request(2) = '1') then
      new_level <= "10";
   eisif(new request(1) = '1') then
     new_level <= "01";
   else
     new_level <= "00";
   end if;
 end process;

 compare:process(current_level_new_level)
 begin

    if(new_level(1) > current_level(1)) then
      should_service <=  '1';
    eisif(new_level(1) < current_level(1)) then
      should_service <= '0';
    eisif(new_level(0) > current_level (0)) then
      should_service <= '1';
    else
     should_service <= '0';
    end if;

  end process;
end;
```

Figure 22



```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
      current_level: in bit_vector(1 downto 0);

      should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin

  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    eisif(new_request(2) = '1') then
      new_level <= "10";
   eisif(new request(1) = '1') then
      new_level <= "01";
   else
      new_level <= "00";
   end if;
 end process;

 compare:process(current_level_new_level)
 begin

    if(new_level(1) > current_level(1)) then
      should_service <=  '1';
    eisif(new_level(1) < current_level(1)) then
      should_service <= '0';
    eisif(new_level(0) > current_level (0)) then
      should_service <= '1';
    else
      should_service <= '0';
    end if;

  end process;
end;
```

Figure 23

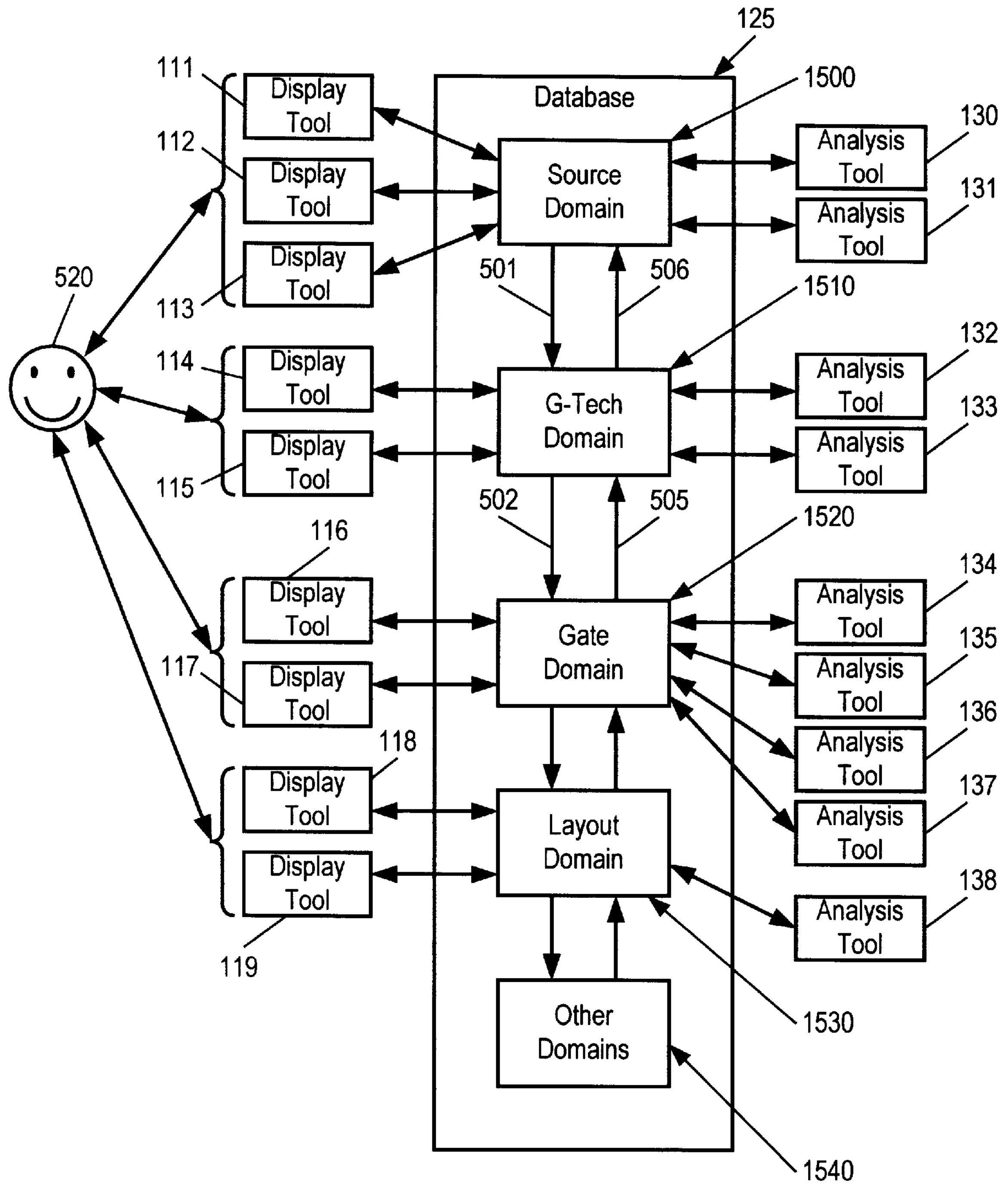

125
Database
1500
111
Display Tool
112
Display Tool
113
Display Tool
Source Domain
130
Analysis Tool
131
Analysis Tool
520
501
506
1510
114
Display Tool
115
Display Tool
G-Tech Domain
132
Analysis Tool
133
Analysis Tool
502
505
1520
116
Display Tool
117
Display Tool
Gate Domain
134
Analysis Tool
135
Analysis Tool
136
Analysis Tool
137
Analysis Tool
118
Display Tool
119
Display Tool
Layout Domain
138
Analysis Tool
Other Domains
1530
1540

Figure 24

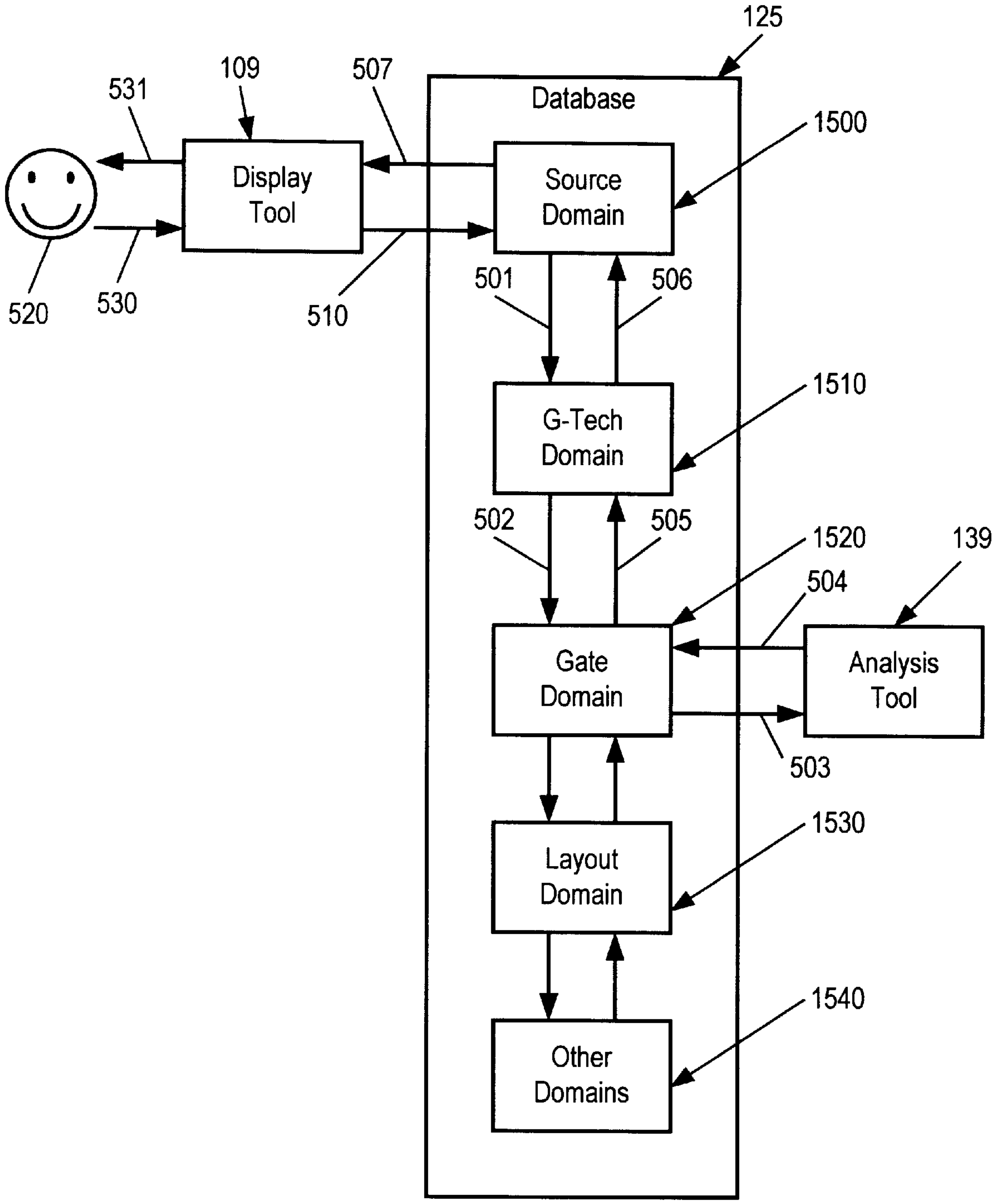
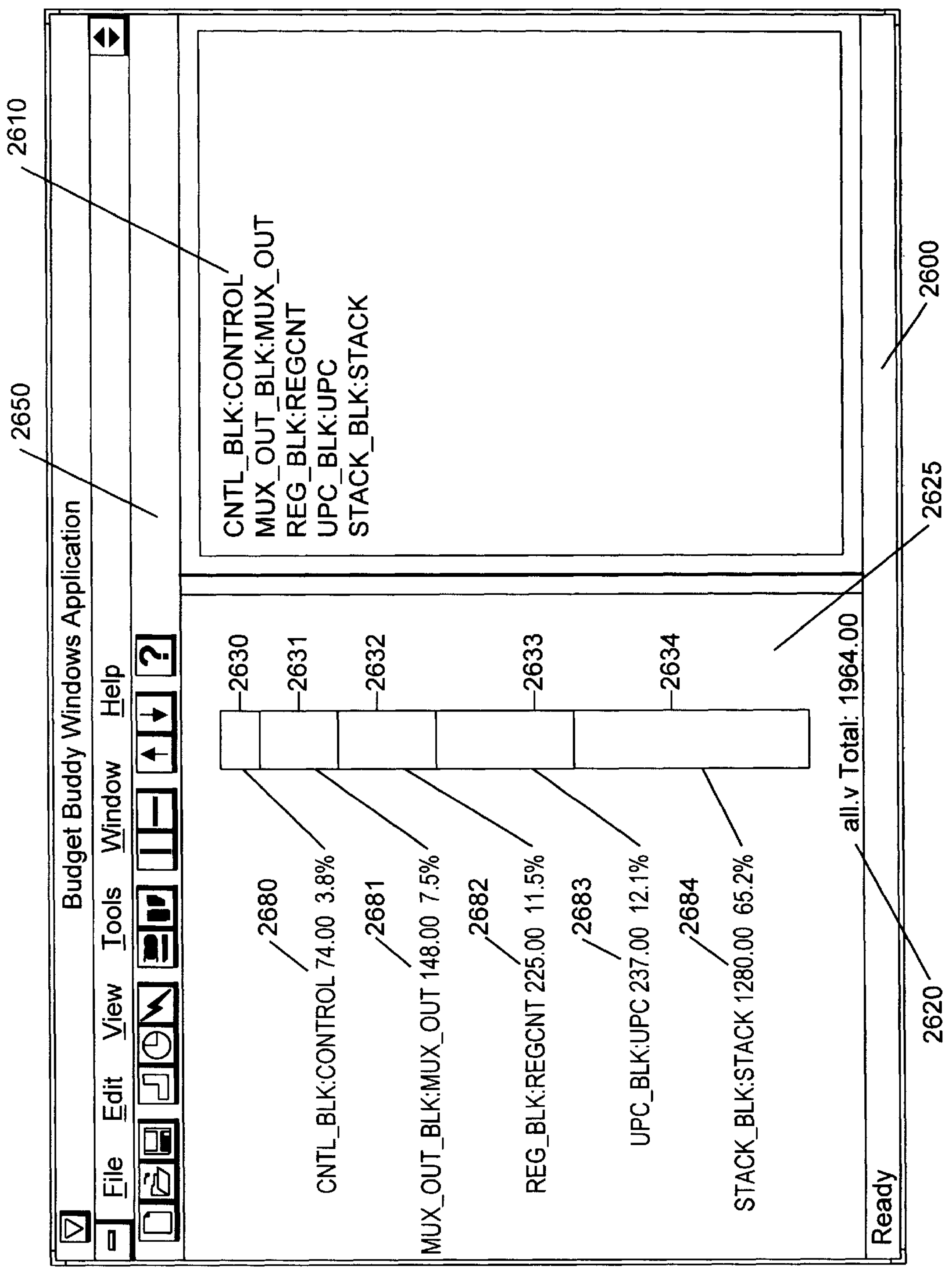

125
Database
109
507
531
Display Tool
Source Domain
1500
520
530
510
501
506
G-Tech Domain
1510
502
505
1520
139
504
Gate Domain
Analysis Tool
503
Layout Domain
1530
Other Domains
1540

Figure 25

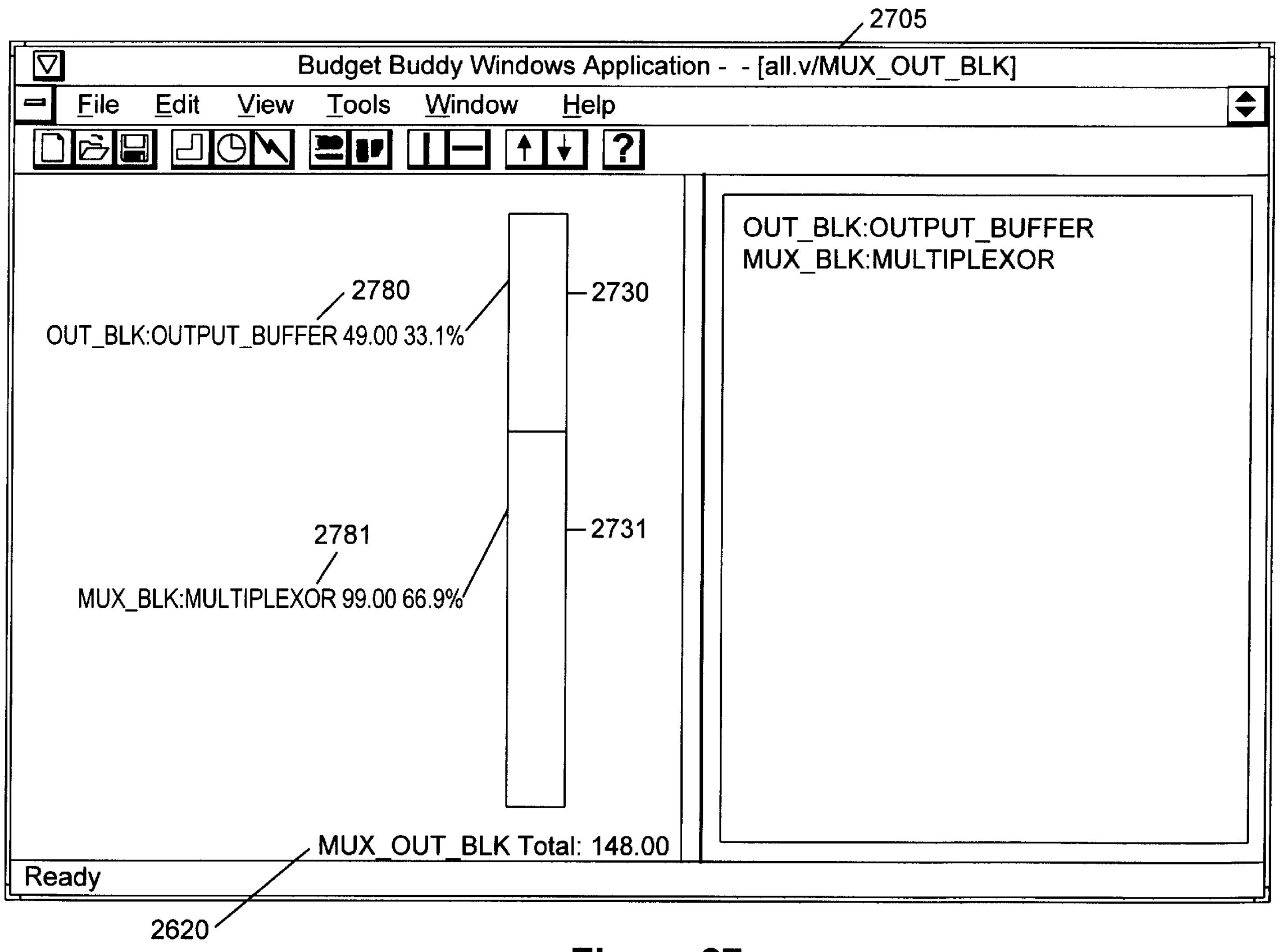


CNTL_BLK:CONTROL
MUX_OUT_BLK:MUX_OUT
REG_BLK:REGCNT
UPC_BLK:UPC
STACK_BLK:STACK

CNTL_BLK:CONTROL 74.00 3.8%
MUX_OUT_BLK:MUX_OUT 148.00 7.5%
REG_BLK:REGCNT 225.00 11.5%
UPC_BLK:UPC 237.00 12.1%
STACK_BLK:STACK 1280.00 65.2%

all.v Total: 1964.00

Figure 26


2705
Budget Buddy Windows Application - - [all.v/MUX_OUT_BLK]
File Edit View Tools Window Help
?
2780
2730
2781
2731
2620
Ready

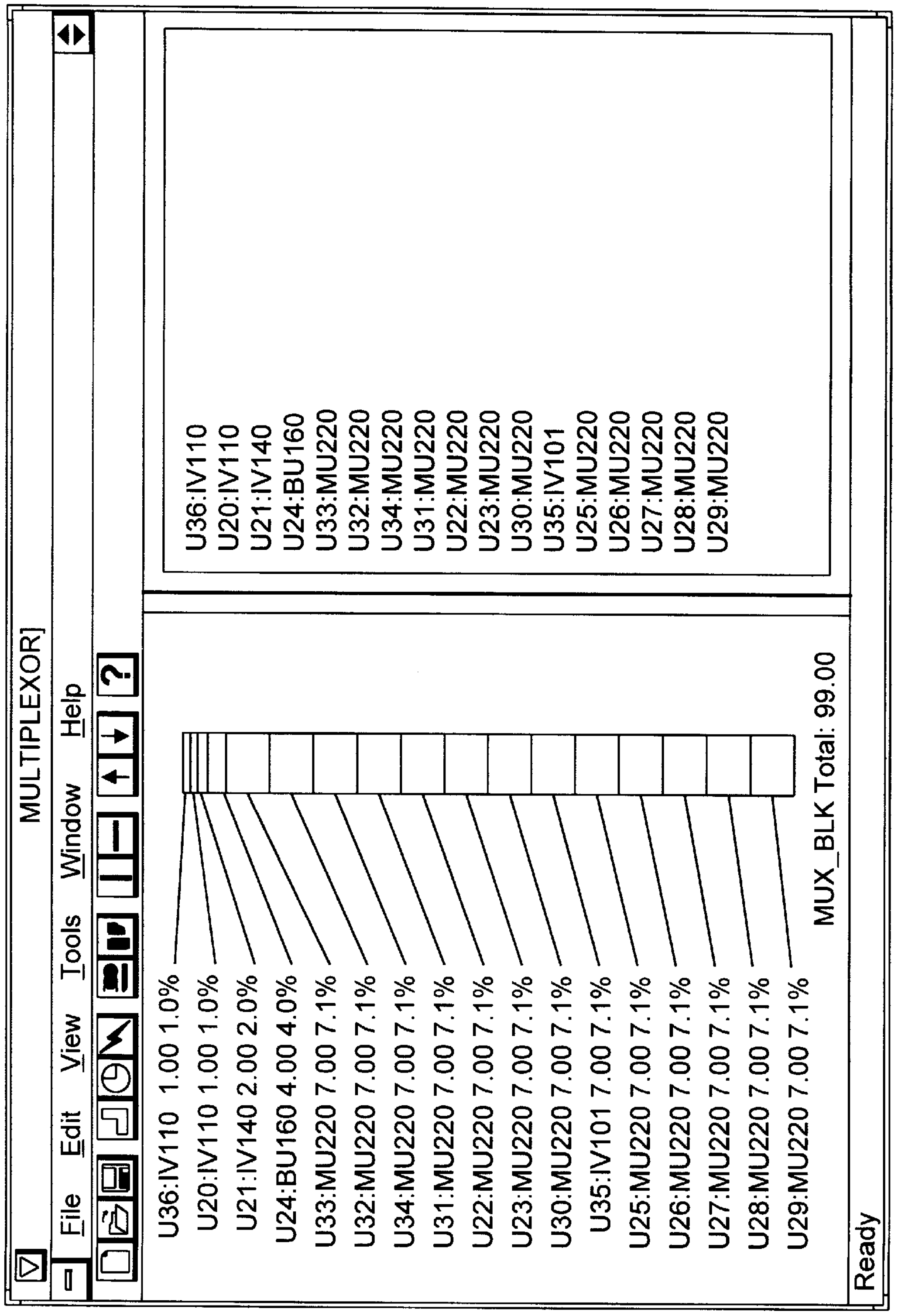
OUT_BLK:OUTPUT_BUFFER 49.00 33.1%

MUX_BLK:MULTIPLEXOR 99.00 66.9%

MUX_OUT_BLK Total: 148.00

OUT_BLK:OUTPUT_BUFFER
MUX_BLK:MULTIPLEXOR

Figure 27

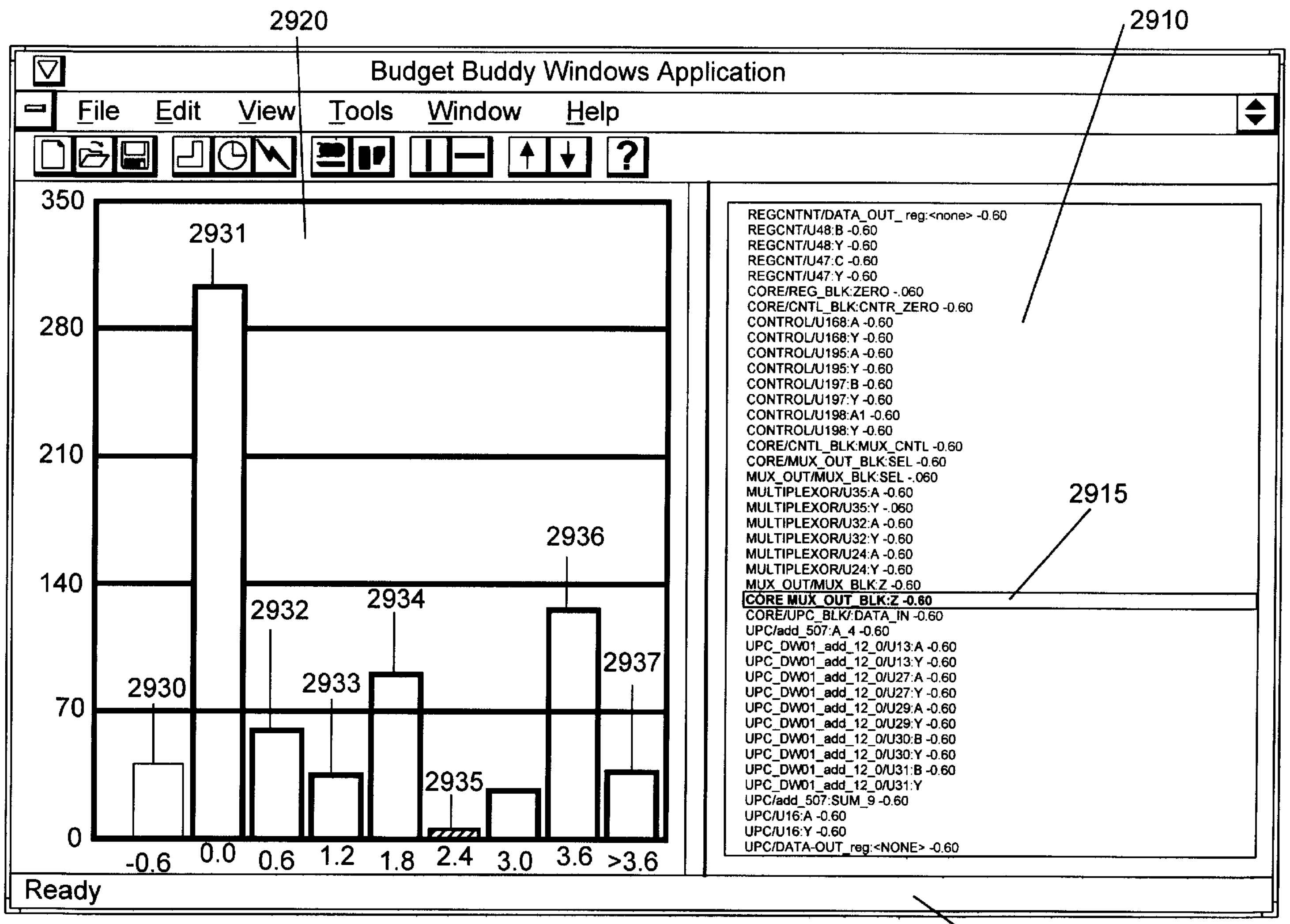


U36:IV110 1.00 1.0%
U20:IV110 1.00 1.0%
U21:IV140 2.00 2.0%
U24:BU160 4.00 4.0%
U33:MU220 7.00 7.1%
U32:MU220 7.00 7.1%
U34:MU220 7.00 7.1%
U31:MU220 7.00 7.1%
U22:MU220 7.00 7.1%
U23:MU220 7.00 7.1%
U30:MU220 7.00 7.1%
U35:IV101 7.00 7.1%
U25:MU220 7.00 7.1%
U26:MU220 7.00 7.1%
U27:MU220 7.00 7.1%
U28:MU220 7.00 7.1%
U29:MU220 7.00 7.1%

MUX_BLK Total: 99.00

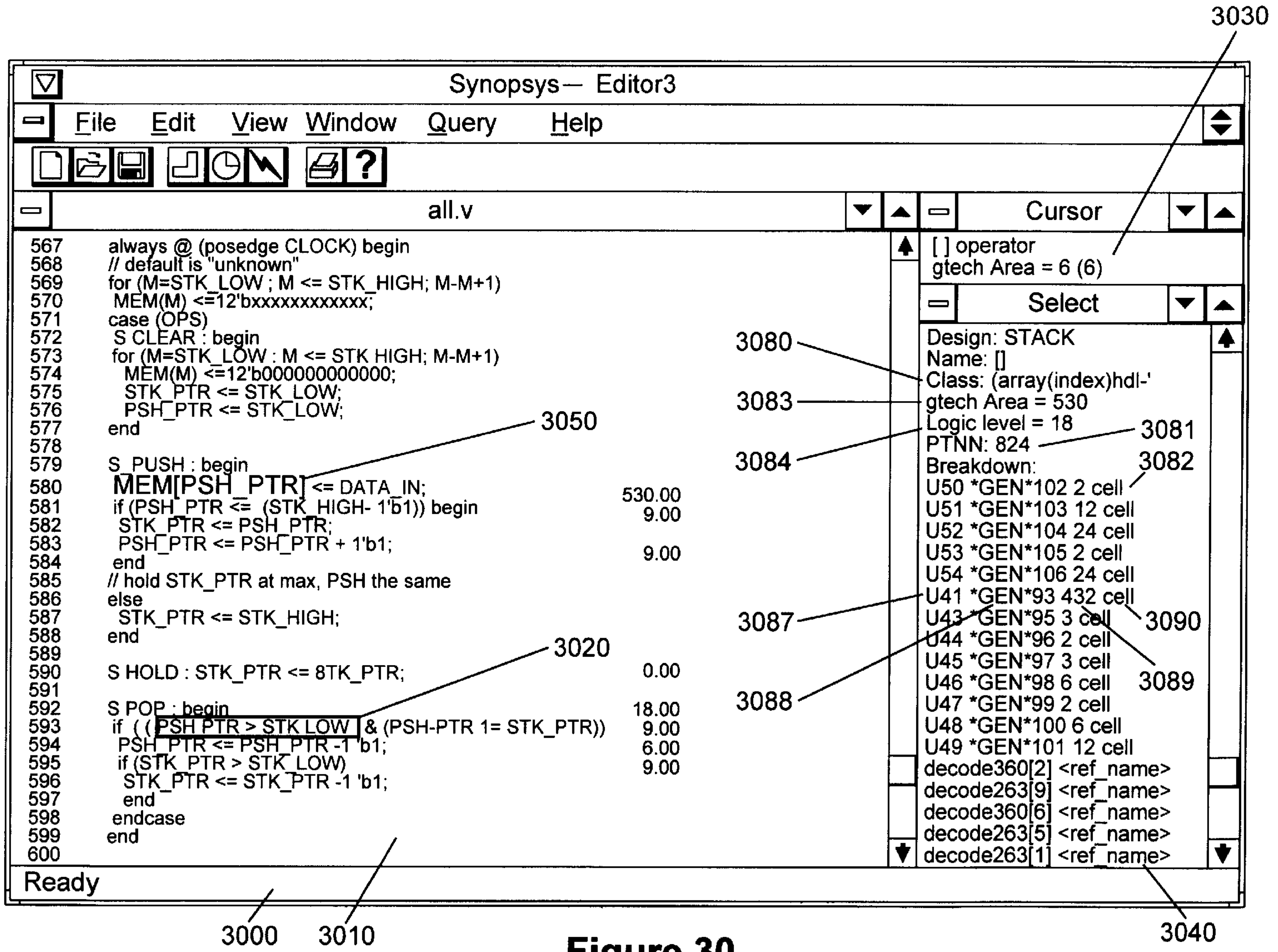
U36:IV110
U20:IV110
U21:IV140
U24:BU160
U33:MU220
U32:MU220
U34:MU220
U31:MU220
U22:MU220
U23:MU220
U30:MU220
U35:IV101
U25:MU220
U26:MU220
U27:MU220
U28:MU220
U29:MU220

Figure 28



REGCNTNT/DATA_OUT_ reg:<none> -0.60
REGCNT/U48:B -0.60
REGCNT/U48:Y -0.60
REGCNT/U47:C -0.60
REGCNT/U47:Y -0.60
CORE/REG_BLK:ZERO -.060
CORE/CNTL_BLK:CNTR_ZERO -0.60
CONTROL/U168:A -0.60
CONTROL/U168:Y -0.60
CONTROL/U195:A -0.60
CONTROL/U195:Y -0.60
CONTROL/U197:B -0.60
CONTROL/U197:Y -0.60
CONTROL/U198:A1 -0.60
CONTROL/U198:Y -0.60
CORE/CNTL_BLK:MUX_CNTL -0.60
CORE/MUX_OUT_BLK:SEL -0.60
MUX_OUT/MUX_BLK:SEL -.060
MULTIPLEXOR/U35:A -0.60
MULTIPLEXOR/U35:Y -.060
MULTIPLEXOR/U32:A -0.60
MULTIPLEXOR/U32:Y -0.60
MULTIPLEXOR/U24:A -0.60
MULTIPLEXOR/U24:Y -0.60
MUX_OUT/MUX_BLK:Z -0.60
CORE MUX_OUT_BLK:Z -0.60
CORE/UPC_BLK/:DATA_IN -0.60
UPC/add_507:A_4 -0.60
UPC_DW01_add_12_0/U13:A -0.60
UPC_DW01_add_12_0/U13:Y -0.60
UPC_DW01_add_12_0/U27:A -0.60
UPC_DW01_add_12_0/U27:Y -0.60
UPC_DW01_add_12_0/U29:A -0.60
UPC_DW01_add_12_0/U29:Y -0.60
UPC_DW01_add_12_0/U30:B -0.60
UPC_DW01_add_12_0/U30:Y -0.60
UPC_DW01_add_12_0/U31:B -0.60
UPC_DW01_add_12_0/U31:Y
UPC/add_507:SUM_9 -0.60
UPC/U16:A -0.60
UPC/U16:Y -0.60
UPC/DATA-OUT_reg:<NONE> -0.60

Figure 29

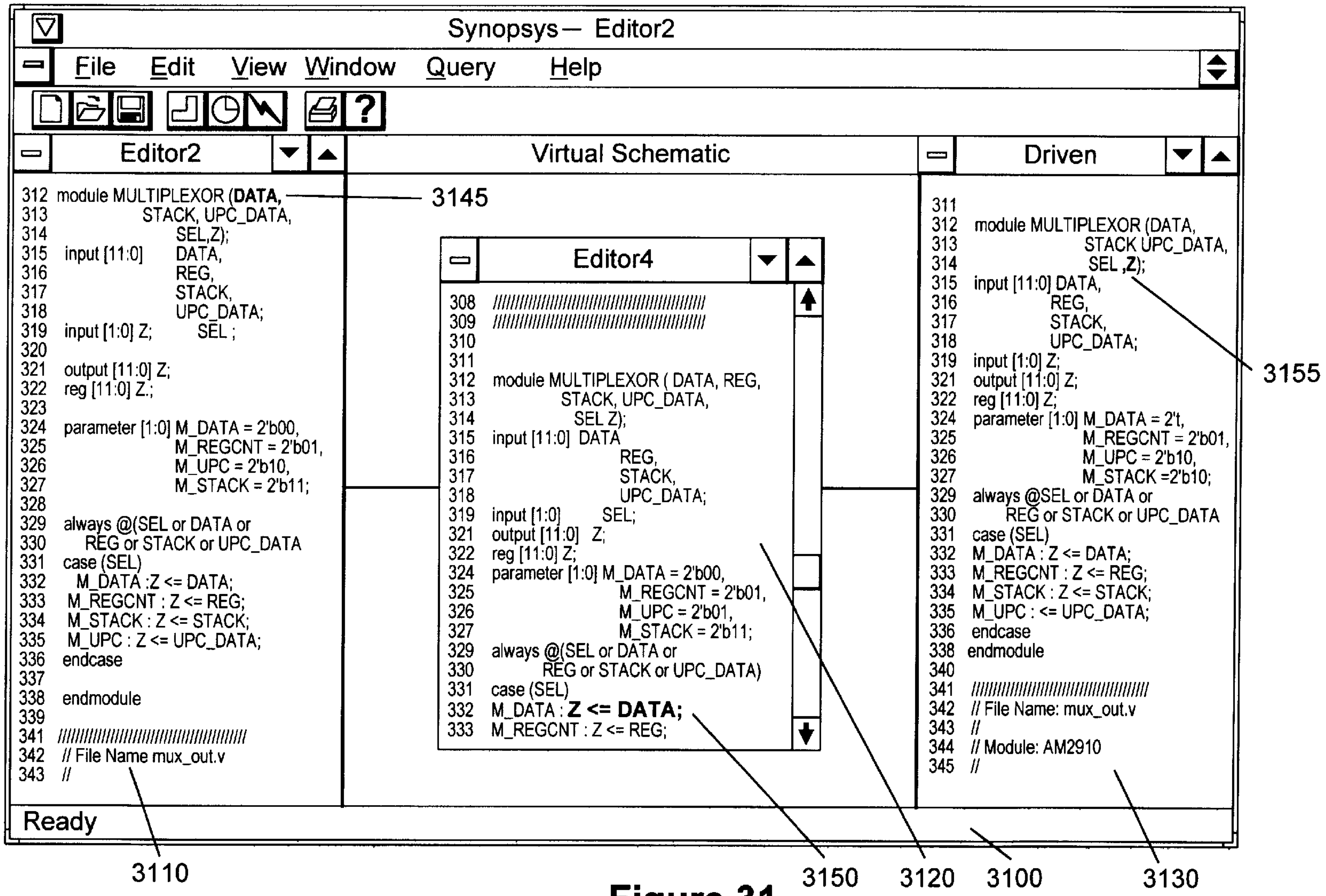


```
always @ (posedge CLOCK) begin
// default is "unknown"
for (M=STK_LOW ; M <= STK_HIGH; M-M+1)
 MEM(M) <=12'bxxxxxxxxxxxx;
case (OPS)
 S CLEAR : begin
 for (M=STK_LOW : M <= STK HIGH; M-M+1)
   MEM(M) <=12'b000000000000;
   STK_PTR <= STK_LOW;
   PSH_PTR <= STK_LOW;
end

S_PUSH : begin
 MEM[PSH_PTR] <= DATA_IN;                          530.00
 if (PSH_PTR <=  (STK_HIGH- 1'b1)) begin             9.00
  STK_PTR <= PSH_PTR;
  PSH_PTR <= PSH_PTR + 1'b1;                         9.00
 end
// hold STK_PTR at max, PSH the same
else
  STK_PTR <= STK_HIGH;
end

S HOLD : STK_PTR <= 8TK_PTR;                         0.00

S POP : begin                                       18.00
 if ( ( PSH PTR > STK LOW ) & (PSH-PTR 1= STK_PTR))  9.00
  PSH_PTR <= PSH_PTR -1 'b1;                          6.00
  if (STK_PTR > STK_LOW)                              9.00
   STK_PTR <= STK_PTR -1 'b1;
   end
 endcase
end
```

```
[ ] operator
gtech Area = 6 (6)
```

```
Design: STACK
Name: []
Class: (array(index)hdl-'
gtech Area = 530
Logic level = 18
PTNN: 824
Breakdown:
U50 *GEN*102 2 cell
U51 *GEN*103 12 cell
U52 *GEN*104 24 cell
U53 *GEN*105 2 cell
U54 *GEN*106 24 cell
U41 *GEN*93 432 cell
U43 *GEN*95 3 cell
U44 *GEN*96 2 cell
U45 *GEN*97 3 cell
U46 *GEN*98 6 cell
U47 *GEN*99 2 cell
U48 *GEN*100 6 cell
U49 *GEN*101 12 cell
decode360[2] <ref_name>
decode263[9] <ref_name>
decode360[6] <ref_name>
decode263[5] <ref_name>
decode263[1] <ref_name>
```

Figure 30

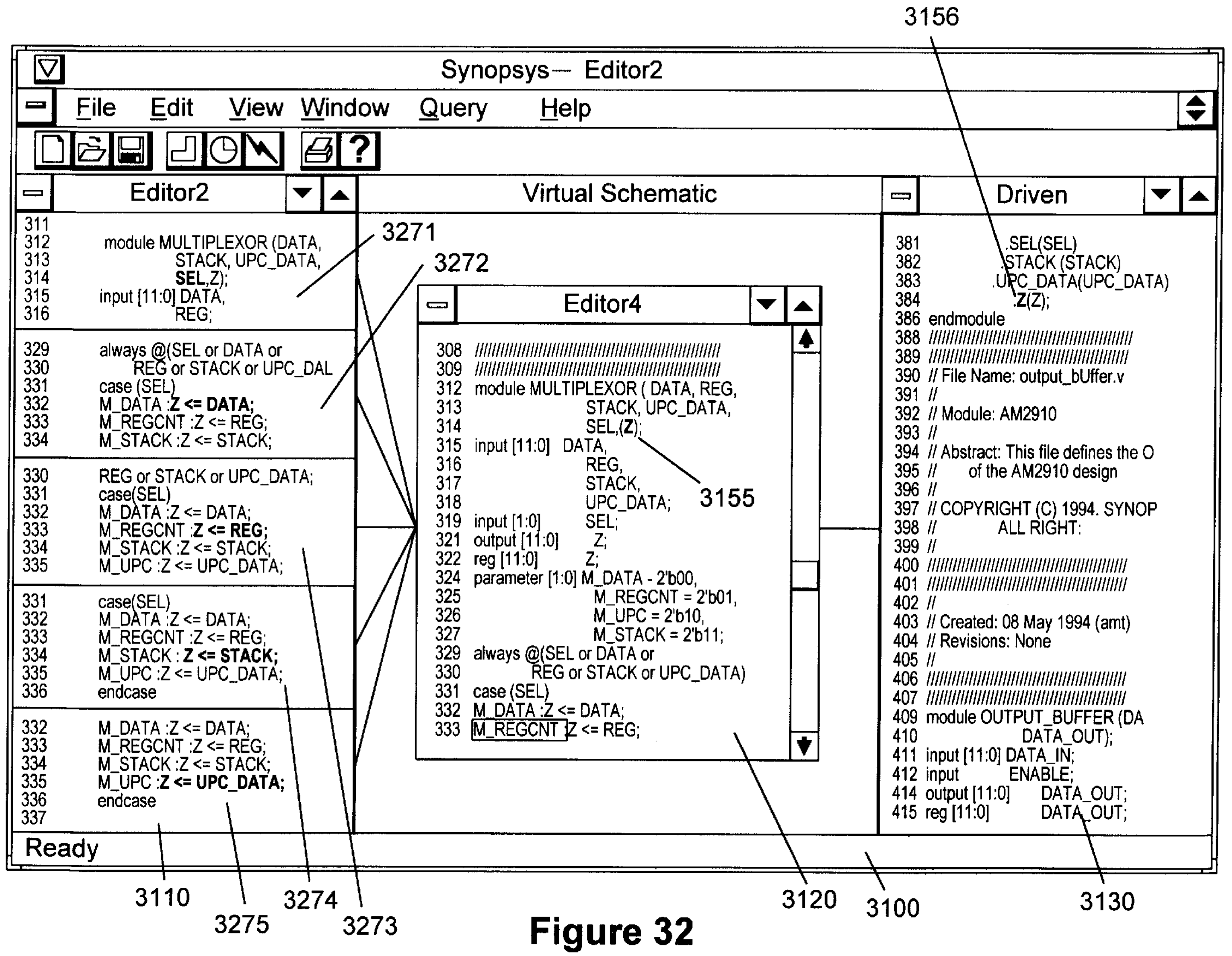


```
module MULTIPLEXOR (DATA,
            STACK, UPC_DATA,
                SEL,Z);
 input [11:0]       DATA,
                REG,
                STACK,
                UPC_DATA;
 input [1:0] Z;       SEL ;

 output [11:0] Z;
 reg [11:0] Z.;

 parameter [1:0] M_DATA = 2'b00,
                M_REGCNT = 2'b01,
                M_UPC = 2'b10,
                M_STACK = 2'b11;

 always @(SEL or DATA or
    REG or STACK or UPC_DATA
 case (SEL)
   M_DATA :Z <= DATA;
  M_REGCNT : Z <= REG;
  M_STACK : Z <= STACK;
  M_UPC : Z <= UPC_DATA;
 endcase

endmodule

/////////////////////////////////////
// File Name mux_out.v
//
```

```
//////////////////////////////////////////
//////////////////////////////////////////


module MULTIPLEXOR ( DATA, REG,
          STACK, UPC_DATA,
            SEL Z);
input [11:0]  DATA
                REG,
                STACK,
                UPC_DATA;
input [1:0]       SEL;
output [11:0]   Z;
reg [11:0] Z;
parameter [1:0] M_DATA = 2'b00,
                M_REGCNT = 2'b01,
                M_UPC = 2'b01,
                M_STACK = 2'b11;
always @(SEL or DATA or
        REG or STACK or UPC_DATA)
case (SEL)
M_DATA : Z <= DATA;
M_REGCNT : Z <= REG;
```

```

module MULTIPLEXOR (DATA,
                STACK UPC_DATA,
                SEL ,Z);
input [11:0] DATA,
            REG,
            STACK,
            UPC_DATA;
input [1:0] Z;
output [11:0] Z;
reg [11:0] Z;
parameter [1:0] M_DATA = 2't,
                M_REGCNT = 2'b01,
                M_UPC = 2'b10,
                M_STACK =2'b10;
always @SEL or DATA or
    REG or STACK or UPC_DATA
case (SEL)
M_DATA : Z <= DATA;
M_REGCNT : Z <= REG;
M_STACK : Z <= STACK;
M_UPC : <= UPC_DATA;
endcase
endmodule

/////////////////////////////////////
// File Name: mux_out.v
//
// Module: AM2910
//
```

Figure 31



```
        module MULTIPLEXOR (DATA,
                STACK, UPC_DATA,
                SEL,Z);
        input [11:0] DATA,
                REG;
```

```
        always @(SEL or DATA or
             REG or STACK or UPC_DAL
        case (SEL)
        M_DATA :Z <= DATA;
        M_REGCNT :Z <= REG;
        M_STACK :Z <= STACK;
```

```
        REG or STACK or UPC_DATA;
        case(SEL)
        M_DATA :Z <= DATA;
        M_REGCNT :Z <= REG;
        M_STACK :Z <= STACK;
        M_UPC :Z <= UPC_DATA;
```

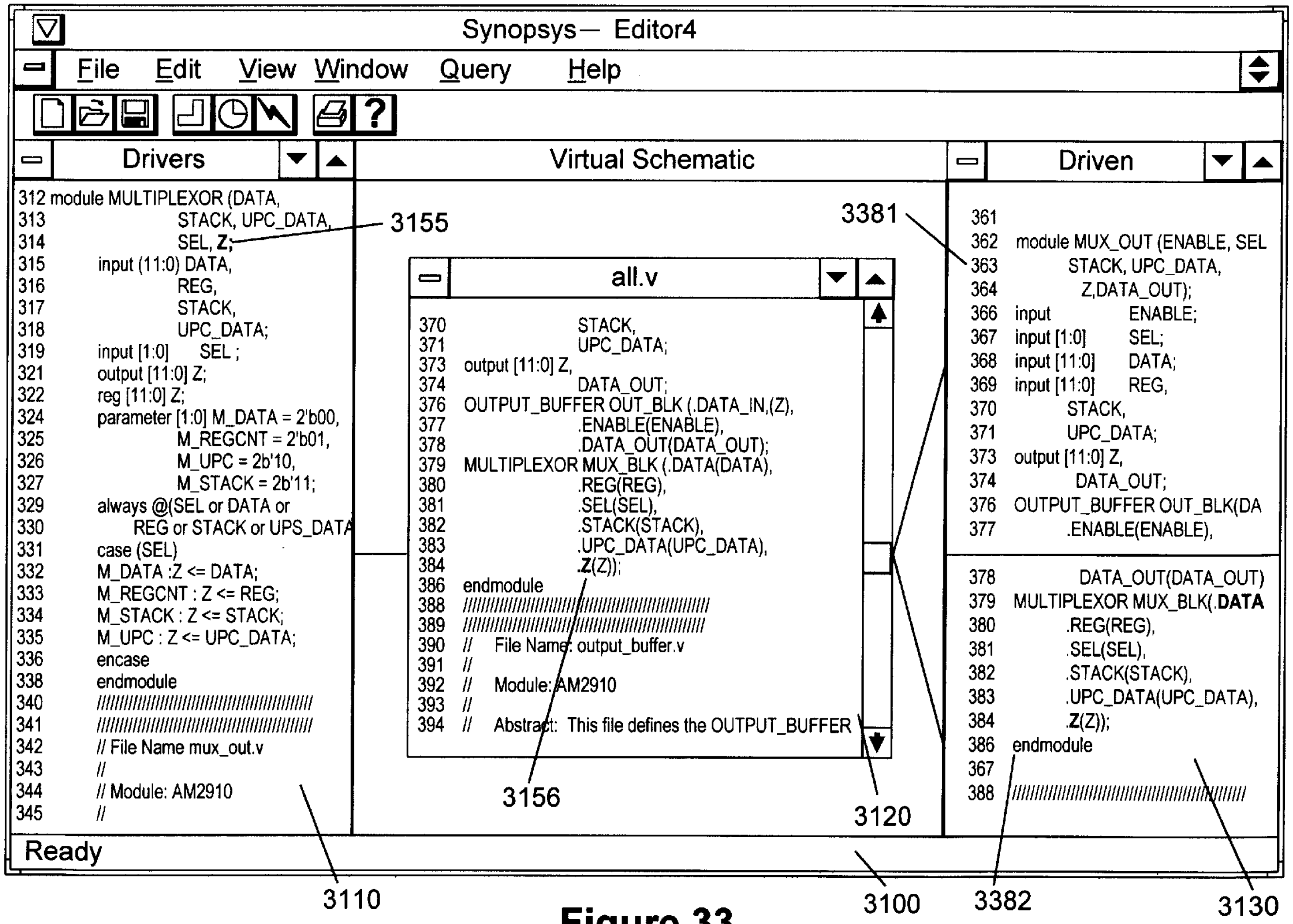
```
        case(SEL)
        M_DATA :Z <= DATA;
        M_REGCNT :Z <= REG;
        M_STACK : Z <= STACK;
        M_UPC :Z <= UPC_DATA;
        endcase
```

```
        M_DATA :Z <= DATA;
        M_REGCNT :Z <= REG;
        M_STACK :Z <= STACK;
        M_UPC :Z <= UPC_DATA;
        endcase
```

```
////////////////////////////////////////////////
////////////////////////////////////////////////
module MULTIPLEXOR ( DATA, REG,
                    STACK, UPC_DATA,
                    SEL,(Z);
input [11:0]  DATA,
              REG,
              STACK,
              UPC_DATA;
input [1:0]   SEL;
output [11:0]   Z;
reg [11:0]    Z;
parameter [1:0] M_DATA - 2'b00,
                M_REGCNT = 2'b01,
                M_UPC = 2'b10,
                M_STACK = 2'b11;
always @(SEL or DATA or
         REG or STACK or UPC_DATA)
case (SEL)
M_DATA :Z <= DATA;
M_REGCNT :Z <= REG;
```

```
            .SEL(SEL)
            .STACK (STACK)
            .UPC_DATA(UPC_DATA)
            .Z(Z);
endmodule
//////////////////////////////////////////
//////////////////////////////////////////
// File Name: output_bUffer.v
//
// Module: AM2910
//
// Abstract: This file defines the O
//      of the AM2910 design
//
// COPYRIGHT (C) 1994. SYNOP
//           ALL RIGHT:
//
//////////////////////////////////////////
//////////////////////////////////////////
//
// Created: 08 May 1994 (amt)
// Revisions: None
//
//////////////////////////////////////////
//////////////////////////////////////////
module OUTPUT_BUFFER (DA
            DATA_OUT);
input [11:0] DATA_IN;
input        ENABLE;
output [11:0]     DATA_OUT;
reg [11:0]        DATA_OUT;
```

Figure 32



```
module MULTIPLEXOR (DATA,
                STACK, UPC_DATA,
                SEL, Z;
      input (11:0) DATA,
                REG,
                STACK,
                UPC_DATA;
      input [1:0]     SEL ;
      output [11:0] Z;
      reg [11:0] Z;
      parameter [1:0] M_DATA = 2'b00,
                M_REGCNT = 2'b01,
                M_UPC = 2b'10,
                M_STACK = 2b'11;
      always @(SEL or DATA or
           REG or STACK or UPS_DATA
      case (SEL)
      M_DATA :Z <= DATA;
      M_REGCNT : Z <= REG;
      M_STACK : Z <= STACK;
      M_UPC : Z <= UPC_DATA;
      encase
      endmodule
      ///////////////////////////////////////////
      ///////////////////////////////////////////
      // File Name mux_out.v
      //
      // Module: AM2910
      //
```

```
                        STACK,
                        UPC_DATA;
output [11:0] Z,
                        DATA_OUT;
OUTPUT_BUFFER OUT_BLK (.DATA_IN,(Z),
                        .ENABLE(ENABLE),
                        .DATA_OUT(DATA_OUT);
MULTIPLEXOR MUX_BLK (.DATA(DATA),
                        .REG(REG),
                        .SEL(SEL),
                        .STACK(STACK),
                        .UPC_DATA(UPC_DATA),
                        .Z(Z));
endmodule
//////////////////////////////////////////////////
//////////////////////////////////////////////////
//    File Name: output_buffer.v
//
//    Module: AM2910
//
//    Abstract:  This file defines the OUTPUT_BUFFER
```

```

module MUX_OUT (ENABLE, SEL
        STACK, UPC_DATA,
          Z,DATA_OUT);
input          ENABLE;
input [1:0]    SEL;
input [11:0]   DATA;
input [11:0]   REG,
        STACK,
        UPC_DATA;
output [11:0] Z,
          DATA_OUT;
OUTPUT_BUFFER OUT_BLK(DA
        .ENABLE(ENABLE),
```

```
          DATA_OUT(DATA_OUT)
MULTIPLEXOR MUX_BLK(.DATA
        .REG(REG),
        .SEL(SEL),
        .STACK(STACK),
        .UPC_DATA(UPC_DATA),
        .Z(Z));
endmodule

///////////////////////////////////////////
```

Figure 33

ARCHITECTURE AND METHODS FOR A HARDWARE DESCRIPTION LANGUAGE SOURCE LEVEL DEBUGGING SYSTEM

RELATED APPLICATION

This application is a continuation-in-part of U.S. application Ser. No. 08/226,147 by Gregory et al, filed on Apr. 12, 1994, now abandoned.

BACKGROUND

1. Field of the Invention

This invention relates to the field of computer aided design for digital circuits, and particularly to debugging digital circuits constructed using logic or behavioral synthesis. This invention also relates to displaying the results of circuit analysis determined in one domain, such as an annotated netlist of gates, in the context of the circuit structure in another domain, such as the hardware description language (HDL) source text.

2. Statement of the Related Art

A digital circuit designer needs to ensure that the circuit performs the correct function subject to many design constraints. For example, the circuit should perform the correct computation in the proper amount of time. The area that the circuit occupies on a semiconductor die should remain within certain bounds. The power that the circuit consumes while operating should also remain within specified bounds. To be economically manufacturable, the circuit should be testable. An economically useful circuit should not take too long to design, manufacture, test or use.

The digital circuit design process involves translating the designer's sometimes ambiguous thoughts about the function and constraints into the tooling necessary to produce a working circuit. For example, producing a full-custom semiconductor chip requires producing masks that define the deposition of chemicals into a substrate as well as producing test patterns that exercise the final product. As another example of tooling, producing a field programmable gate array requires generating the bit pattern to be downloaded into the chip to specify the configuration of the architecture. Computer Aided Design (CAD) tools facilitate the iterative translation of the designer's developing thoughts into the tooling required to produce a working circuit that satisfies the design constraints. The process of iteratively adjusting a design to meet its requirements is called debugging.

The historical model of the digital design process using conventional CAD tools for a semi-conductor chip is as follows. The designer first conceives of a particular function to implement, as well as constraints such as timing or area that the implementation must meet. Next, historically, the designer mentally transforms the desired function into a high level circuit consisting of components such as gates, adders, registers and RAMS. The designer then captures that insight by drawing a schematic of a circuit that implements that function with a schematic capture tool. The schematic shows how more primitive functional elements, such as gates or transistors, connect together to form more sophisticated functions such as arithmetic logic units. In addition, modern schematic capture tools allow the designer to divide the design hierarchically into interconnected pieces, and then allow the user to specify the details of each of the pieces separately. For example, Design Architect by Mentor Graphics of Wilsonville, Oregon provides these schematic capture functions.

Conventional CAD tools, such as those indicated above, can then take the connections in the schematic and other

information to evaluate the circuit and to specify the tooling necessary to construct the circuit. Such tools evaluate the circuit in many ways. For example, commercial CAD tools often have a simulator that predicts the response of the circuit to designer specified input patterns. QuickSim II by Mentor Graphics of Wilsonville, Oreg. is a commonly used simulator. Another common CAD tool is a path delay analyzer that identifies the longest timing path in a circuit design. DesignTime by Synopsys, Inc. of Mountain View, Calif. is a tool that provides path delay analysis.

Conventional CAD tools also have the ability to generate the geometric layout of the circuit with layout tools. Cell3 Ensemble by Cadence of San Jose, Calif. is an example of this type of tool. Layout tools are required to produce masks to make a semi-conductor chip.

Conventional CAD tools also have the ability to check that the circuit meets the design rules, and to identify the location of any errors to the designer. Design rules help ensure that the specified circuit will operate once manufactured.

Conventional CAD tools also are used to determine how testable a circuit is, and to generate test patterns automatically. Showing the designer the parts of the circuit that are not testable allows the designer to make modifications that will increase the probability of making a successful chip or circuit. Generating test patterns automatically allows for more thorough testing of the circuit immediately after manufacturing.

As described above, the concept of debugging a circuit design historically refers to the process by which a circuit designer specified a particular implementation with a schematic capture tool, and then used various circuit evaluation tools to verify that the implementation did what the circuit designer wanted. For example, the designer would use a simulator to determine if the circuit produced appropriate outputs from specified inputs. The designer could use the path delay analyzer to determine whether the current design was fast enough to meet the requirements. The layout tools could inform the designer whether the design would fit in the available space.

When a particular design did not meet the designer's expectations or requirements, the designer then modified the design. For example, if the circuit was too slow, the designer identified the offending circuitry and revamped it to increase performance. If the circuit was too large, then the designer revised the circuit to use fewer or smaller components. If the circuit did not behave as required, the designer changed the components and the interconnections to produce the correct function. Because the conventional CAD tools began the analysis with the captured circuit, the timing or area problems could be readily identified to the designer. Because the designer specified the structure of the circuit, the designer could then thoughtfully make adjustments. However, because historically the designer mentally performed the transformation from desired function to circuit, the CAD tools were limited in their ability to identify functional problems to the designer.

Logic synthesis developed to provide the designer with an automatic mechanism to translate a hardware description language (HDL) description of a desired function to a structural description of a circuit that performed the desired function. Logic synthesis begins with the designer describing the desired function using VHDL, Verilog, or any other logic synthesis source language, to specify the behavior. A translator then converts that description into gates and other circuit structures that directly correspond statement by statement with the designer's description. Theoretically, conventional CAD tools could then evaluate the resulting circuit and develop the appropriate tooling.

However, the circuit created by a statement-by-statement translation is generally large and slow. In logic synthesis, translation is followed by logic optimization. Optimization replaces the directly translated structure with a functionally equivalent, yet improved structure.

Unfortunately, the circuit transformations performed by the logic optimizer usually modifies the structure of the circuit. This results in a circuit that is unrecognizable by the designer. The fact that the designer generally can not recognize the original function performed by the transformed circuit makes debugging synthesized logic difficult. Conventional evaluation tools can determine the timing or area problems in the transformed circuit, but the designer can not relate those problems easily to the HDL source specification. Theoretically, the designer could manually determine what part of the HDL specification caused the problem. With that insight, the designer could make the desired changes at the HDL specification, and resynthesize the entire circuit. If the designer's problem occurred in a part of the circuit that passed through the optimizer with few changes, manual backtracking might work. However, the optimization process generally makes many changes, making it either difficult or impossible to backtrack because many points in the original circuit do not exist in the final circuit.

Alternatively, the designer could simply modify the final circuit directly. This would not allow the designer to resynthesize the design from the HDL specification because the designer's circuit changes would be overwritten by subsequent translation and optimization steps. This would destroy the value gained by using the synthesis approach to design.

There are some existing tools and techniques for determining where and how to modify a HDL source specification. One tool allows the designer to examine a gate in the optimized circuit schematic, and inquire where that gate came from in the HDL source, provided that it directly existed in the source. If the tool could not tell where a gate came from, it would say so. An example of a gate tracing tool is Design Analyzer's source-to-gates function, produced by Synopsys, Inc. of Mountain View, Calif. However, many gates are removed and others are added during the translation and optimization process. The gate tracing tool has not proven to be very useful in helping designers debug circuits because many of the components in a optimized circuit can not be traced back to the HDL source specification.

Another method of debugging a synthesized circuit is to partition the design into hierarchical components, and synthesize and optimize the smaller pieces. Because the synthesis and optimization tools generally do not traverse primary inputs and outputs, such a partitioning can reduce the size of the backtracking problem. However, it has the disadvantage that the designer may have to rewrite functionally correct, but nonetheless problematic, synthesis source code to isolate the troublesome parts of the circuit. In addition, this approach will greatly limit the optimizer's ability to reduce the area and increase the speed of the resulting circuits because the optimizer will be constrained by the designer's partition.

In addition, a designer can be mislead by the results obtained by debugging by partitioning. The designer's bug in the circuit might be that it is too slow or too big. Partitioning the synthesis source to locate the cause will result in a different circuit than the unpartitioned source. Therefore, the problem that the designer is chasing could vanish or be made significantly worse by the debugging process itself.

Conventionally, using a synthesizer to translate a specification into a circut can also raise substantial computational problems to incorporate minor changes into a design late in the design process. For example, a designer could have the design fairly close to completion when the designer discovers the need to make a small functional change, such as inverting a particular signal. Intuitively, one would expect that such a small change would require only a small change in the circuit all the way to the layout level. However, it is quite possible that, with conventional synthesis and optimization tools, a small change could require substantial changes in the circuit and the layout. Currently, a designer can limit this kind of problem by partitioning the design into smaller pieces and thus limiting the effect to the directly implicated pieces. However, as described previously, inappropriate or unduly narrow partitioning can limit the ability of the optimization tools to construct a good circuit.

3. A Conventional Design and Debugging Process Overview

FIG. **1** shows an overview of the conventional process for designing and debugging circuits specified with a Hardware Description Language (HDL). The process begins with the designer writing HDL source code **100**. A typical language used for specifying circuits is VHDL which is described in the IEEE Standard VHDL Language Reference Manual available from the Institute of Electrical and Electronic Engineers in Piscataway, N.J., which is hereby incorporated by reference. VHDL stands for Very high speed integrated circuit Hardware Description Language. Another language used for specifying circuits is Verilog that is described in Hardware Modeling with Verilog HDL by Eliezer Sternheim, Rajvir Singh, and Yatin Trivedi, published by Automata Publishing Company, Palo Alto, Calif., 1990, which is hereby incorporated by reference. Verilog is also described in the Verilog Hardware Description Language Reference Manual (LRM), version 1.0, November 1991, which is published by Open Verilog International, and is hereby incorporated by reference. The examples used in this document are in VHDL, but the principles readily apply to other circuit specification languages.

After writing a HDL description of a desired function, the designer then simulates the function **101** embedded in the description with a HDL simulator. An example of a functional simulator is VHDL System Simulator that is available from Synopsys, Inc. of Mountain View, Calif. The functional simulator allows the designer to determine whether the circuit produces correct values in response to inputs without regard to timing, area or power constraints. A functional simulator can perform function-only simulation relatively quickly, thus enabling the designer to determine that the circuit will produce the desired output.

If there is a problem with the function, the designer can fix function problems **102** by examining the simulation output and going back to writing HDL code **100**. Functional simulation executes the source specification directly without generating or optimizing circuits. Therefore, problems identified during functional simulation can readily be linked to their cause in the HDL source.

If the designer believes that the described circuit provides the correct function, then designer then specifies constraints for the synthesis process **103**, e.g. maximum clocking periods, total circuit area, and maximum power. This part of the process is described in Design Compiler Family Reference Manual, Version 3.1a, which is available from Synopsys, Inc. of Mountain View, Calif. and is hereby incorporated by reference. Examples of Computer Aided

Design software that use constraint specification are Synergy by Cadence, and Autologic by Mentor Graphics, and Design Compiler by Synopsys.

After developing constraints, the designer then proceeds to synthesize **104** a circuit from the HDL description produced in the writing HDL **100** step. This step involves translating the description into an initial circuit that correspond directly with the statements in the source HDL. An example of software that performs this function is described in the VHDL Compiler Reference Manual, Version 3.1a, which is available from Synopsys, and is hereby incorporated by reference. After translation, the initial circuit is then optimized into a final circuit that meets the performance constraints established in step **103**. Prior to optimization, it is a straight-forward task to identify which circuit element of the initial circuit corresponds to what part of the HDL source code. Conventionally, because of the extensive manipulations performed during the optimization process, such identification after optimization becomes almost impossible except at registers and module interface boundaries.

FIG. **2** shows the intermediate data structures involved in the synthesis process **104**. The synthesis process begins with HDL source **900**. The translator creates a data structure called a parse tree **901** that represents the organizational structure of the HDL. The translator then turns the parse tree into an initial circuit **902**. Russ B. Segal's Master's Thesis, "BDSYN: Logic Design Translator" at the University of California at Berkeley, Memo#UCB/ERL M87/33, describes a such a translator, and is hereby incorporated by reference. United States patent application Ser. No. 07/632,439, filed on Dec. 21, 1990, entitled "Method and Apparatus for Synthesizing HDL Descriptions with Conditional Assignments" by Gregory et al, and commonly assigned to Synopsys, Inc. also describes such a translator, and is hereby incorporated by reference. An example of a tool that does this is version 3.1a of the HDL compiler available from Synopsys, Inc.

An optimizer is used to produce the final circuit **903** from the initial circuit **902**. The optimization process is explained in "Logic Synthesis Through Local Transformations" by J. Darringer, W. Joyner, L. Berman, and L. Trevillyan in the IBM Journal of Research and Development, volume 25, number 4, July 1981, pages 272–280, which is hereby incorporated by reference. It is also explained in "LSS: A System for Production Logic Synthesis" by J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L. Trevillyan in the IBM Journal of Research and Development, volume 28, number 5, September 1984, pages 537–545, which is hereby incorporated by reference. It is also explained in "MIS: A Multiple-Level Logic Optimization System" by R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang in the IEEE Transactions on Computer Aided Design, Volume 6, number 6, November 1987, pages 1062–1081, which is hereby incorporated by reference. It is also explained in the Ph.D. dissertation "Logic Synthesis for VLSI Design" by R. Rudell at the University of California at Berkeley in 1989, which is hereby incorporated by reference. The optimization process is also described in the Design Compiler Family Reference Manual, version 3.1a which is available from Synopsys, and is hereby incorporated by reference. An example of software that performs this function is the Design Compiler available from Synopsys, Inc. Other examples of software that performs optimization include Synergy from Cadence, Inc., and AutoLogic by Mentor Graphics.

One approach to optimization is to group one or more initial circuit elements together, and replace those elements

with a functionally equivalent collection of elements that has better characteristics than the collection of elements replaced. This results in an intermediate circuit that is functionally equivalent to the original. This intermediate circuit can then some or all of its elements grouped for another replacement. This process is repeated until the optimizer meets the constraints imposed in step **103** of FIG. **1**, or is unable to make any further improvement.

Before beginning the optimization process, the components of the initial circuit can be divided into two groups: those components that must be preserved through the optimization process, and those that can be replaced with functional equivalents. For example, a logic optimizer may replace a block of boolean logic with another block so long as function is maintained. Generally, replaceable components can also be eliminated. Examples of components that are generally preserved through the optimization process are primary inputs, primary outputs and registers.

After developing a circuit, the designer can then analyze the circuit **105** using conventional analysis tools, as shown in FIG. **1**. For example, the designer could estimate the area that the circuit consumes or what the longest delay path is in the circuit. This analysis can identify problems to the designer. The analysis report **904** is often a text document, as shown in FIG. **2**.

After identifying timing, area, testing or power problems with the analysis tools, the designer then devises a means to adjust the circuit to fix these problems **108**. Ideally, the designer would go back to the HDL where the function is specified and make adjustments there. However, because it is currently hard to identify the specific places in the source HDL that led to the problem, modifying the appropriate part of the HDL is currently not an effective debugging technique. The designer can usually identify which hierarchical module contains some, of the problem, and the designer could then manually rewrite that module to create more primary inputs and outputs to examine. This is very time consuming and is generally done as a last resort. Alternatively, the designer could adjust the constraints **103** and synthesizes the circuit **104** again to see if the problem is alleviated.

After debugging the circuit, the designer then releases the design for fabrication **106**.

4. System Performance

In addition to the debugging problems presented by the transformations made by the logic synthesis process, there are also difficulties associated with efficiently and economically constructing CAD systems that compute and display analysis results. Conceptually, after specifying a design, debugging a circuit involves having the designer repeatedly (1) determine a particular circuit characteristic or property that the designer wants to know about, (2) identify a kind of analysis that will provide information about that characteristic, (3) instruct the CAD system to perform that analysis, (4) display the results of that analysis, and (5) gain insight into the desired characteristic from the display. The designer is interested in completing these steps as quickly as possible. Circuit CAD tools have historically facilitated this goal by making the instruction and display steps computationally efficient. To improve response times, circuit CAD tools have often tightly couple the software that performed the analysis to the software that performed the display function. This was often done by having the display software depend heavily on the data structure used to process or store the results of the analysis.

For example, timing analysis often reveals the portions of the circuit that are too slow. Reviewing this analysis historically has involved examining the schematic and tracing the critical path. However, as described previously, the schematic may have little to do with the designer's specification of the circuit.

This intimate linking of display to analysis causes several problems. First, as described, in conjunction with logic synthesis, displaying the data structure linked with the analysis may not be particularly insightful to the designer. Second, such an intimate linking requires more software development and designer training. If there were N kinds of displays, and M kinds of analysis, and each kind of display could be combined with each kind of analysis, then conventional CAD systems tended to have N*M individual display/analysis programs. This requires the designer to become proficient with a multiplicity of slightly different interfaces as well as requiring the tool supplier to construct all of these tools.

However, modern CAD tools must support the development of chips containing millions of transistors. Designing chips with this many components requires that the CAD tools display and analyze large data structures. Processing large data structures tends to reduce the response time of CAD tools. The conventional technique of tightly coupling the display software to the analysis software helps the CAD system maintain a reasonable response time with the large data structures.

Intimately linking the display tools to the analysis tool data structures poses some problems. First, it tends to require the maker of the CAD tool to produce a large number of products that require support. Second, the variety of such tools tends to introduce variations in the command interface that the designer must use to identify and initiate circuit analysis. Both of these problems lead to frustrated designers and tool builders.

5. Background Conclusion

Using HDL synthesis can simplify the task of circuit design by allowing the designer to specify the required function in an HDL textual description without specifying the details of the circuit implementation. After creating a circuit using synthesis, the designer can use conventional circuit analysis tools to determine characteristics of the final circuit. Conventional analysis will describe such things as the area consumed by different parts of the circuit, or what the longest delay path is through the circuit. Using these analysis results, the designer can then identify which portions of the circuit are problematic. However, because the optimization portion of synthesis often transforms the design substantially, it is difficult, if not impossible, except in certain special cases, to relate specific portions of the final circuit to the HDL source that generated those portions. This inability to trace the circuit analysis results easily to the HDL source represents a substantial barrier for debugging circuits efficiently.

SUMMARY OF THE INVENTION

The present invention provides a method for displaying the results of synthesized circuit analysis visually near the HDL source specification that generated the circuit. Circuit analysis provides information about the characteristics of each portion of the synthesized circuit. The present invention relates the analysis results of each portion of the synthesized circuit to the particular part of the HDL specification that generated that circuit portion. This permits the designer to modify the part of the HDL specification that is responsible for problems identified by circuit analysis. The synthesis process works by translating HDL source code into

an initial circuit. Each point in the initial circuit corresponds directly with a particular construct in the HDL source. A final, more efficient circuit is constructed from the initial circuit by logic optimization. Connecting the results of the analysis to the source requires identifying points in the final circuit that be traced directly to the initial circuit. Circuit analysis results corresponding to these invariant points in the circuit can therefore be directly related to the appropriate part of the HDL source, and thus can be displayed near that part.

The present invention provides a method for introducing additional points in the design that remain traceable through the optimization process without requiring reorganization or modification of the HDL source. The present invention provides these additional points, for example, by artificially injecting primary inputs or outputs into the initial circuit, and noting where in the HDL source these points came from.

The present invention provides a method for linking information gleaned from evaluating and analyzing a synthesized circuit to the source code that produced the circuit. The present invention establishes the link by providing a designer with the ability to mark the synthesis source code in the places that the designer wants to be able to debug. The designer marks the source code with a particular text phrase, such as "probe", along with some additional optional information. During translation, the translator generates a circuit the provides the same function as it did without the "probe" statement, but adds additional information or components to the initial circuit that indicate that certain components should not be replaced during optimization. Because those components will not be replaced during optimization, the circuit analysis results corresponding to any unreplaced components that are in the final circuit will be directly and traceably related to those components in the initial circuit. Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore be displayed near the appropriate portion of the HDL. Allowing for the interjection of unreplaced components by the designer facilitates debugging without rewriting the designer's original hierarchical design or manually backtracking through the optimization process.

In another aspect of the invention, the designer can assign a priority level to each probe to help manage the debugging process. These priority levels could then be used to activate or deactivate selected probes as a group. An activated probe would establish a link through the synthesis process to facilitate debugging. An inactive probe would have no effect on the synthesis process, and would not establish a debugging link. Establishing many links would provide the designer with a large degree of debugging information, but could limit the ability of the synthesis process to provide a good circuit. Establishing too few links may not provide enough guidance to the designer to resolve circuit problems. By selectively activating groups of probes at different times during the debugging process, the designer can analyze different portions of the design without the probes themselves unduly interfering in the process.

By providing a facility for displaying the results of circuit analysis near the HDL that created the circuit, the present invention allows a designer to make more effective use of logic synthesis and reduce the complexity of the circuit debugging process.

An aspect of the present invention provides a method and system for processing requests from designers about the characteristics associated with the logic synthesis source specifying a circuit, and displaying the results of circuit analysis with a consistent set of display tools that are not intimately tied to the data structure used for the circuit analysis. The present invention achieves this by first observing that the representation of the circuit can be divided into domains characterized by the structure of the information used to represent the design. Then the tool builder can develop domain dependent display tools for examining the state of the design in that domain. In addition, the tool builder can also develop tools that evaluate or analyze the state of the circuit in a particular domain. Display tools showing the circuit structure in one domain can obtain information related to analysis obtained in another domain by the forward and backward linkages provided by this invention.

The designer can inquire about the characteristics related to a specific part of the design by first examining part of the design in one domain with a display tool. This domain is the inquiry domain. After identifying a relevant portion of the design in the inquiry domain, the designer selects a constituent piece of the design to evaluate, and makes an inquiry about that piece. This information constitutes a query. The display tool forwards the identification of the object in the inquiry domain and an identifier indicating the requested analysis or evaluation to the database. The database then determines the domain that would contain the relevant analysis results. If those results do not yet exist, the database invokes the appropriate analysis tool to put those results into the database. Using the linkage established with the HDL-debugging method, the database locates the related object in the analysis domain. From the related object, the appropriate information is passed back to the display tool where the designer can see it displayed appropriately.

One aspect of the present invention provides display tools that are not dependent on the structure of the domain in which the analysis is actually performed. Another aspect of the invention provides analysis tools that are not dependent on the structure of the display domain. Another aspect of the invention is to allow the different display and analysis tools to remain independent from one another.

BRIEF DESCRIPTION OF THE DRAWINGS

DETAILED DESCRIPTION OF THE INVENTION

The present invention comprises a novel method for the using the links and establishing new links between an HDL source description of a circuit and the analysis results of the translated and optimized actual circuit that arises from the source description. The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the preferred embodiment will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the invention. Thus, the present invention is not intended to be limited to the embodiment shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

Improved Design and Debugging Process

FIG. **3** shows the general design and debugging process in accordance with the present invention. The designer writes HDL with probe statements **150**. The probe statements identify the places in the resulting circuit that the designer will wish to examine. The designer may not initially know where probes will be required until later in the design process. The probes have no impact on functionality so functional simulation **101** and functional repair **102** proceed as before. The designer also constrains synthesis **103** as before.

Synthesizing with probes **154** differs from conventional synthesis **104**. When the translator encounters a probe statement, the translator interjects information into the netlist to indicate to the optimizer that optimization should not proceed "past" or "through" that particular point. The optimizer then produces a new circuit subject to these constraints. In one embodiment, the probed portions of the HDL source are treated as both primary inputs and primary outputs during translation and optimization.

The circuit analysis step **105** proceeds as before. After analysis, the tool then uses information developed during translation to relate the results of the analysis to the HDL source as indicated by step **120**.

With the information gleaned from the probes, the designer can now identify problems and evaluate solutions that directly change the HDL, as shown in step **121**.

After completely analyzing and debugging the design, the actual circuit can be fabricated as it was before as shown in step **106**.

As described above, optimizers divide an initial circuit's components into two groups: components that can be replaced and those that can't. One method of implementing probes would be to mark the components of the initial circuit attributable to the probed part of the HDL as components that can not be replaced during optimization. For example, the translator could create a primary input and a primary output at the places in the initial circuit corresponding to the probed part of the HDL source.

After optimization, the circuit can then be analyzed with conventional circuit analysis tools to provide an analysis report **904**. For example, the analysis report can indicate the time it takes for a signal to reach various points in the optimized circuit. Analysis could also reveal what portions of the design are not testable or occupy a large amount of area. Because each report provides information linked to the nodes of the optimized circuit, each report therefore provides information about each node that did not change during the optimizer's iterations. Each node that did not change during the optimization process can be traced to a particular place in the HDL source using the information developed in constructing the parse tree.

Examples of Using and Processing Probes

FIG. **4** through FIG. **18** illustrate by example how probes work and how the debugging information could be displayed to the user. The examples use VHDL as the source language. The principles illustrated do not depend on the particular language.

FIG. **4** shows a text editor window **300** containing a VHDL code fragment **400** that does not contain any probe statements. The code fragment shown in FIG. **4** is repeated below:

```
if (C and B) then
        Z <= not(A or B);
else
        Z <= not(B);
end if;
```

FIG. **5** shows a graphical representation of the parse tree constructed while translating the source code in FIG. **4**. Link **1100** is the connection that this fragment has with the rest of the VHDL description. The "if" statement in VHDL has three parts: a condition; a VHDL statement to process when the condition is true; and a VHDL statement to process when the condition is false. The condition is dealt with in the tree descending from node **1001**. The true condition is handled by the tree descending from node **1004**. The false condition is handled by the tree descending from node **1010**. The assignments represented by nodes **1004** and **1010** are used to link the signal values represented by node **1005** and node **1011** to their functions represented in the trees descending from nodes **1006** and **1012** respectively.

Without a probe statement, the VHDL fragment would translate into the circuit of FIG. **6**. Inputs A, B, and C are schematically represented by the connectors **200**, **201**, and **202**. The "if" statement would translate into multiplexor **231**. The condition "(C and B)" would translate into and gate **232**. The "true" condition would translate into nor gate **233** while the "false" condition would translate into invertor **230**.

FIG. **7** shows a circuit optimized from the digital circuit of FIG. **6**. In particular, the logic function that this code fragment really performs is not(B). At this point, without probes in the conventional synthesis design process, the designer can not obtain much information about the internal timing information descending from the fragment. For example, if the designer needed to know when the value of not(A or B) was computed to help analyze some other aspect of the design, then the designer would not be able to deduce that information from the resulting analysis.

FIG. **8** shows how a probe statement **401** could be inserted into the source description. The code fragment is repeated below:

```
if (C and B) then
        Z <= not(A or B); --Synopsys probe_statement
else
        Z <= not(B);
end if;
```

In VHDL, "--" begins a comment. The word "Synopsys" immediately after the "--" indicates that this is not an ordinary comment, but rather a directive to the translator or other part of the computer aided design tool environment. The word "probe_statement" indicates that this particular VHDL statement should be processed so that it will be possible to relate analysis information to this point in the circuit.

FIG. **9** shows a circuit that a translator could produce from the code fragment shown in FIG. **8** with the probe statement. The parse tree produced with the probe statement is the same as before, namely the tree of FIG. **5**. However, the probe statement will cause the signal represented by node **1005** to behave as both a primary output and a primary input.

In creating the circuit of FIG. **9** from the parse tree, the translator could add temporary input **203** and a new temporary output **221**. In addition to synthesizing this circuit, the translator connects the new temporary input to the new output at a higher level in the net list produced from translating the whole specification.

FIG. **10** shows the circuit of FIG. **9** after optimization. The optimizer is not permitted to optimize circuits past the boundaries established by the probe statement. This means that nor gate **233** of FIG. **9** would be transformed into nor gate **253** of the optimized circuit. The nor gates are not necessarily identical because additional details may be specified about the gates during the design compilation process. Those details are not relevant to the implementation of probes.

Because the logic optimization process left temporary input **203** and temporary output **221** alone, and those points correspond to a particular point in the HDL circuit, any analytic result related to temporary input **203** or temporary output **221** can be identified with the probe statement **401** in the HDL. FIG. **11** shows how timing information could be related back to the HDL in a special text window **301**. For example, suppose a critical path analysis tool determined that it took 1.0 nanoseconds to produce temporary output **221** of FIG. **10** after a clock edge arrived at a flip-flop somewhere else in the circuit. By using the fact that temporary output **221** came from this line of the source, the timing result **500** can be displayed next to the appropriate line of the output.

FIG. **12** through FIG. **19** show another way to use probes to evaluate the performance of blocks of HDL code. FIG. **12** shows a text window with an HDL entity described. This text has no probe statements inserted, and the code is repeated below.

```
entity interrupt__controller is
     port(new__request : in bit__vector(3 downto 1);
               current__level: in bit__vector(1 downto 0);
               should__service: out bit);
end;
architecture synthesizable of interrupt__controller is
     signal new__level: bit__vector(1 downto 0);
begin
     decode: process(new__request)
     begin
          if(new__request(3) = '1') then
               new__level <= "11";
          elsif(new__request(2) = '1') then
               new__level <= "10";
          elsif(new__request(1) = '1') then
               new__level <= "01";
          else
               new__level <= "00";
          end if;
     end process;
     compare: process(current__level, new__level)
     begin
          if(new__level(1) > current__level(i)) then
               should__service <= '1';
          elsif(new__level(1) < current__level(1)) then
               should__service <= '0';
          elsif(new__level(0) > current__level(__)) then
               should__service <= '1';
          else
               should__service <= '0';
          end if;
     end process;
end;
```

This VHDL source code computes whether a new interrupt should be serviced. It determines what the new level of the interrupt is by determining what input line is asserted, and comparing the interrupt priority level associated with that with the level of the currently pending interrupt. If the new level is higher, then the output should__service is set high, and otherwise it is set low.

FIG. **13** shows the circuit resulting from translating the VHDL source. FIG. **14** shows the circuit that results from optimizing the circuit in FIG. **13**.

FIG. **15** shows a special text window that summarizes some of the performance information about the circuit. The analysis tool can provide information about the design as viewed from the inputs. An analysis tool could provide an estimate of the area of the design by counting gates or compute the longest delay through the entire design. The designer might conclude that this circuit is too big or too slow. Because all of the inner details of the design have been optimized, it is not realistic for the designer to examine the schematic in FIG. **14** manually and determine whether the decoding function or the comparing function is too big or too slow.

To determine where the problems lie, the designer could insert block probes textually near the definition of the decoding and comparing processes, as shown in the text window of FIG. **16**. This would probe all signals entering or leaving the sequence of HDL code delineated by the begin block and end block statements. When translated, the probed HDL would become the circuit of FIG. **17**. The translator would create temporary inputs **2010** and **2011**, and temporary outputs **2000** and **2001**, much as it did with the statement probe.

The optimizer transforms the circuit of FIG. **17** into the circuit of FIG. **18**. This allows the analysis tools to compute timing and area characteristics of both parts of the circuit. A special purpose display tool can then display, for example, timing and area analysis, as shown in FIG. **19**. In this example, the decoding circuit is approximately the same size and delay as the comparator circuit.

The block probes used in FIGS. **16** through **20** illustrate a mechanism to select many parts of the HDL source for probing without typing a text statement probe command for each line of HDL. In particular, the block probes in the previous example prohibited the optimizer from eliminating the inputs and outputs of the block. Another kind of block probe that might be useful is one that places a probe on every signal within the block. This will significantly limit the optimizer's ability to improve the circuit within the block, but it will give the designer the maximum amount of information about how the HDL source was translated. A third kind of block probe could place a probe on every statement in the block. A fourth kind of block probe could place a probe on every signal driving multiplexor control inputs within the block. Multiplexors are generated in HDL translation to implement "if" and "case" constructs. Probing the control inputs to these multiplexors would provide a designer with much insight into the circuit behavior.

As described previously, the circuit produced when probes are used can be different from the circuit that occurs when probes are not used. Because probes interfere with the ability of the optimizer to produce higher quality circuits, the designer generally should only use them when the designer needs to gather particular information. During the debugging process, a designer may insert many probe statements into the HDL source at various times to discover the characteristics of different parts of the circuit. One method of managing the number and location of active probes would be to let the designer specify a number with every probe statement that would be the statement's priority. When invoking the optimizer, the designer could then specify a optimization priority. The optimizer would then eliminate every probe with a priority greater than the invoked priority. This would have the effect of permitting the designer to only use the most important probes without having to textually remove the lesser probes.

Alternate Methods for Probing

FIG. **20** shows an alternate method of having the translator implement probes. The circuit in FIG. **20** would come from the VHDL code fragment of FIG. **11**. Instead of creating both a new input and a new output, only a new output **210** is created.

FIG. **21** shows another method of having the translator implement probes. The circuit in FIG. **21** would come from the VHDL code fragment of FIG. **11**. Here, a new part **270** is inserted into the design. This part would need to have an additional attribute associated with it to indicate to the optimizer that it was not to be eliminated.

Another method of implementing probes requires attaching an attribute to the net connecting gates together. For example, in processing a probe statement, the translator could create the circuit in FIG. **6**, but add information to net **280** that net **280** could not be removed during the optimization process.

Tracing Transitive Fan-In and Fan-Out in the Source HDL

During debugging, the designer may need to consider the impact that a change in one part of the design would have other parts of the design. If the designer is considering changing a particular function in HDL, the designer would find it useful to identify all of the inputs to that function or all of the outputs to that function. The collection of all of the points in a circuit leading to a particular point is referred to as the Transitive Fan-In of that point. The collection of all of the points in a circuit that depend on the value of a particular point is the transitive fan-out. While tracing all of the inputs (or outputs) of a particular part of the source HDL is a difficult task using only the HDL source, using the direct correspondence between the HDL and the initial circuit formed during translation makes it possible to highlight the inputs (or outputs) in the source HDL.

For example, a designer might be interested in finding all of the computations that feed into the multi-bit signal new_level in the VHDL source of FIG. **12**. The designer could select the signal new_level with a mouse and enter a trace command with keystrokes or mouse clicks. A tracing program would trace the desired signal through all computations leading to primary inputs. As the tracing program traces signals back through the initial circuit, it identifies those points corresponding directly to the HDL source, and makes it possible for the text editor to highlight the corresponding HDL source. FIG. **22** provides an example of the text that would be highlighted during such a trace.

Alternatively, the designer might be interested only in the primary inputs or registers that lead to the indicated signal. FIG. **23** shows the result of tracing new_level back to the primary inputs.

System Architecture

FIG. **24** illustrates the architectural relationship between the display tools **111, 112, 113, 114, 115, 116, 117, 118**, and **119** the analysis tools **130, 131, 132, 133, 134, 135, 136, 137**, and **138** and the database **125**. The designer **520** interacts with the display tools by seeing information on a screen and providing information to the tool by keyboard and mouse. The database **125** contains different representations of the circuit design under development. These representations are grouped into domains **1500, 1510, 1520, 1530**, and **1540**. The analysis tools **130, 131, 132, 133, 134, 135, 136**, and **137** interact with the design information in the various domains to summarize and evaluate the design.

Design Representation: Domains

Providing useful information to the designer about the current state of a circuit design requires an explanation of how different aspects of a designs are stored in a CAD system. Previous sections described the process that a designer goes through to create and debug a design using logic synthesis. In going through the design process, the designer, through the CAD system, manipulated and transformed digital data having some information about the circuit into other digital data that had different information about the circuit. This section explains how the information describing all of the aspects of a design is organized into domains.

A domain is a collection of the design data that contains common structural characteristics. Each domain represents a particular level of abstraction of the design information. In the CAD system architecture of FIG. **24**, the design can be divided into a source domain **1500**, a generic technology domain **1510**, which is also known as a G-Tech domain, a gate domain **1520**, a layout domain **1530**, and possibly other domains **1540**. The design data in one domain can be the result of a transformation of design data from another domain using design tools, such as a logic synthesizer, and libraries of components.

The source domain **1500** contains the HDL source files that the designer creates in step **100** of FIG. **1** or step **150** of FIG. **3**. It also contains the parse trees and symbol tables generated during the translation step of logic synthesis. Here, this portion of the design representation only contains information about the desired function of the circuit without reference to circuit topology or technology.

The generic technology domain **1510** contains the initial circuit that arises from the translation step of the synthesis process, as shown in step **104** of FIG. **1** or step **154** of FIG. **3**. Data stored in the generic technology domain **1510** contains information about the topology of the circuit, but does not have information about the specific technology to be used.

The gate domain **1520** contains the final circuit that arises after the optimization step of the synthesis process, as shown by link **502**. The debugging method previously described establishes the reverse link **505**. Like the generic technology domain **1510**, the data in the gate domain **1520** contains information about how components are connected together. However, in the gate domain, a particular technology is specified, thus providing information about the physical characteristics of the components used to implement the desired function. It is in this domain that preliminary timing, area, power, testability, and other calculations of step **105** of FIG. **1** and FIG. **3** can be made.

The layout domain **1530** contains information about the geometric placement of the components on the chip substrate and the connections between the components. The design information in the layout domain **1530** is obtained from the design information in the gate domain **1520** by using placement and routing tools.

It would also be possible to have additional domains, as shown by other domains **1540**.

Objects within a Domain

Conceptually, the design information within a domain can be thought of as a collection of interconnected objects, with the objects and the connections possessing certain characteristics. For example, in the source domain, the objects could include the text of the HDL source code or the nodes of the parse tree constructed from the source code or the entries in the symbol table. In the gate domain **1520**, the objects could include the individual gates or other library parts or the connections between them. In any case, database **125** maintains the relationships between the objects within the domain. The debugging method presented discussed earlier shows how to establish a link between a group of objects in one domain and a group of related objects in another domain.

Display Tools

The information specifying the design is actually represented in binary form within a computer system. For the human designer **520** to develop and debug the design, information about that design must be presented in a form that can be understood by the human, such as by a visual display on a monitor. In addition, the designer **520** must also have a way to manipulate the design information, such as by keyboard and mouse. Display tools **111, 112, 113, 114, 115, 116, 117, 118**, and **119** provide a mechanism for that interaction.

However, circuit design information is particularly complicated, and efficiently allowing the designer **520** to control and analyze the design requires that the display tools take advantage of the structure of the design data. One way to take advantage of the structure is to design particular display tools to communicate with a particular domain in the database. For example, as shown in FIG. **24**, display tools **111, 112**, and **113** communicate with the source domain **1500**. Display tools **114** and **115** communicate with the G-Tech domain **1510**. Display tools **116** and **117** communicate with the gate domain **1520** while display tools **118** and **119** communicate with the layout domain.

A display tool interacts with its related domain by communicating messages back and forth. These messages involve specifying an object in the domain and some related action to perform on that object. The display tool is customized to interact with a particular domain in the sense that it is constructed to process objects in that domain. In particular, a display tool needs to be able to display an object and have a list available of operations that can be performed on that object, although the display tool does not perform those operations. The database, in contrast, needs to be able to provide the display tool with objects and process requests associated with objects.

Analysis Tools

Analysis tools are used to process information contained within a domain. In general terms, there are two kinds of analysis that one might want to perform in a given domain. The more conventional kind of analysis tool takes one or more objects in a domain, and computes characteristics associated with those objects. For example, in the gate domain, a timing analyzer is used to compute the delay times and slacks to various nodes in the circuit. In the analysis sense, the timing analyzer is determining a certain characteristic, for example, the time that the data signal will become valid after a clock edge, of certain objects in the domain, namely a connection between gates.

Another kind of analysis involves summarizing the characteristics associated with a particular group of objects. For example, estimating the total area in a circuit at the gate level involves summing the area associated with each component and connection.

For an analysis tools to summarize characteristics of objects, the analysis tools requires information about the structure of the objects and the connections between them. Therefore, analysis tools are associated with a particular domain. For example, in FIG. **24**, analysis tools **130** and **131** deal with the source domain **1500**, while analysis tools **132** and **133** deal with the G-Tech domain **1510**. Analysis tools **134, 135, 136**, and **137** deal with the gate domain **1520**, while analysis tool **138** interacts with the layout domain.

In general terms, analysis tools communicate with the database **125** by messages. The database provides the analysis tool with one or more objects, and the analysis tool responds by setting one or more characteristics of those objects or returning a summary of the characteristics of those objects, or both.

Using the Architecture for Debugging with Reference to the Source Domain

This section explains how a designer uses the CAD system architecture to relate characteristics of the design found in one domain to aspects of the design found in another domain. FIG. **25** shows a simplified view of the architecture of FIG. **24** involving only one display tool and one analysis tool. For this example, the design in question is assumed to be complete to the gate domain **1520**.

The designer **520** of FIG. **25** begins the pursuit of design insight by first obtaining a display that engages in the debugging process. Next, he obtains a visual representation of an aspect of the design stored, for example, in the source domain **1500** using display tool **109**. At the time the display tool **109** begins executing, the database **125** provides the display tool **109** with a list of analysis requests that the designer **520** can request. The display tool **109** could display the text of an HDL source file. The HDL text is an object in the source domain **1500**. The processing example described in this section corresponds to the example used in FIG. **12** through FIG. **19**.

The designer **520** then selects an object to evaluate by selecting some text, such as the decode: process statement in the text of FIG. **16**, and allowing the database **125** to identify the parse tree nodes corresponding to the selected text. Alternatively, to improve performance, the database **125** could transfer data to the display tool to allow the display tool to identify the parse tree node. A good technique for relating particular pieces of text with corresponding parts of a parse tree is described in a co-pending application by Gregory entitled "Method and Apparatus for Context Sensitive Displays", filed on Jun. 3, 1994 with express mail certificate TB596163183US, which is hereby incorporated by reference. The parse nodes are also objects in the source domain.

The designer **520** also selects a type of analysis to be performed from the available choices. One approach is that the designer **520** selects it from a menu or a push button. Another approach would be to have the designer **520** select it before selecting the text. In this example, assume that the designer **520** asks for an estimate of the area of the decode process.

The parse node and the desired analysis form a query that is sent to the database **125**. Because area requires technology specific information, the database **125** can identify that the required information is in the gate domain **1520**. In essence, the database **125** then needs to compute the area corresponding to the identified parse node. The database does this by identifying the gates in the gate domain **1520** that correspond to that parse node. One way that this can be done is to use the techniques described earlier using hierarchy and probes. In this example, the designer **520** wanted to know what the area of one of the processes is, and the designer **520** used probe points to segregate the design to permit this.

Given that the database **125** has identified the relevant gates, it then needs to compute the total area. In the gate domain **1520**, each gate is an object with an associated area characteristic. In this situation, the database could calculate the total area with a summarizing type of analysis tool **139** that sums up the area of each gate in a list. In other situations, it may be necessary to perform a sequence of simpler analysis operations on parts of the data structure to deduce the correct result. The analysis request from the display tool could also include a list of such instructions. Analysis tool **139** then produces a number that is handed back to database **125** which then sends to display tool **109** which can display the result directly or use it to modify the display characteristics.

Display Tools

The flexibility of the architecture shown in FIG. **24** permits additional display tools that relate circuit analysis information about parts of the circuit to the source text that generates those parts. FIG. **26** through FIG. **33** show novel display techniques for displaying circuit analysis data. FIGS. **26** through **33** show the display tools displaying an AMD2910A design. The source and circuit is described in Introduction to HDL-Based Design Using VHDL, by Steve Carlson, published in 1991, which is hereby incorporated by reference. This book is available from Synopsys, Inc., 700 East Middlefield Road, Mountain View, Calif. 94043-4033.

Stacked Bar Graph Display

FIG. **26** shows a stacked bar graph displaying information about the relative contributions of parts of the synthesis source. Often a design written in an HDL is described hierarchically, with higher level modules containing lower level modules. At a particular level in the hierarchy, the designer might want to know the characteristics of the modules visible at that level. The stacked bar graph of FIG. **26** shows relative areas associated with different parts of the design. At the highest level of the AMD 2910A, there are five functional sub-blocks: CNTL__BLK, MUX__OUT__BLK, REG__BLK, UPC__BLK, and STACK__BLK. The names of these blocks are shown in the object list area **2610** of window **2600**. The total measured area is displayed at the bottom of the window. The total area of the circuit is shown as 1964.0 gates. In this example, the characteristic is area. However, other characteristics such as power and time can be similarly displayed.

Each of the sub-blocks has a measured characteristic which is shown by text statements **2680** through **2684**. For example, CNTL__BLK uses 74.00 gates, which is 3.8% of the total as shown by statement **2680**. MUX__OUT__BLK occupies 148.00 gates, which is 7.5% of the total as shown by statement **2681**. REG__BLK occupies 225.00 gates, which is 11.5% of the total as shown by statement **2682**. UPC__BLK occupies **237.00** gates, which is 12.1% of the total as shown by statement **2683**. STACK__BLK occupies 1280.00 gates, which is 65.2% of the total as shown by statement **2684**. A stacked bar graph is constructed by drawing a graphical box corresponding to each functional sub-block with the size of the box proportional to the percentage of the sub-block's characteristic to the total characteristic. This is shown with boxes **2630** through **2634**.

Importantly, the stacked bar graph display of FIG. **26** can be constructed without reference to the physical nature of the particular characteristic. Therefore, power or timing can be displayed as easily as area. The database need only transfer the names of objects and the numerical value associated with those objects to the display tool.

Furthermore, each box, such as box **2630** through box **2634**, forming part of the stacked bar graph can also be a selectable button. The user can "push" the button and gain information about the sub-block associated with the box. FIG. **27** shows the result of the user selecting the sub-block MUX__OUT__BLK box by selecting box **2631**. Here, the sub-block MUX__OUT__BLK itself contains two sub-blocks, OUT__BLK and MUX__BLK. The total measured characteristic **2620** changes to 148.00 to reflect the size of MUX__OUT__BLK. The sub-block OUT__BLK has 49.00 gates representing 33.1% of the area of MUX__OUT__BLK, as indicated by statement **2780**. This information is also shown graphically by box **2730**. The sub-block MUX__BLK has 99.00 gates representing 66.9% of the area of MUX__OUT__BLK, as indicated by statement **2781**. This information is also shown graphically by box **2731**. In addition, the window also shows the current location in the hierarchy with a path statement **2705**. In addition, statements such as statement **2780** could also act as buttons to change levels.

FIG. **28** shows the information displayed if the designer selects MUX__BLK to see how the 99.00 gates are allocated.

Histogram Display

The histogram display of FIG. **29** can provide a designer with information about the extent of problems currently in the design. For example, one aspect of the performance of a digital circuit is the delay along any path from the output of one flip-flop to the input of another. Once a designer specifies the clock waveforms, a timing verifier can determine arrival and required times throughout the clocked circuit. At any point in the circuit, the arrival time can be determined relative to a clock edge by measuring the longest path from a register affected by the clock to the point within the circuit. Similarly, required times may be computed relative to a clock edge by measuring the longest path from the point in the circuit to a register affected by the clock. Once the relationships between all clocks and all clock waveforms are specified, the timing verifier can determine the worst slack for each point within the circuit by subtracting arrival from required times for each possible combination of relevant clock edges. The smallest, or most negative result can be considered the worst slack for that point in the circuit. The Design Compiler Reference Manual V3.1a from Synopsys contains more information about timing analysis, and is hereby incorporated by reference. If any node has a negative slack time, then the timing goal has not been met. If only a few nodes are have negative slacks, or the negative slacks are close to zero, then the designer may merely have a relatively small problem that can be fixed by tuning a small portion of the design. However, if many nodes have negative slacks or the slack times are large in magnitude, then the designer may be facing a substantial design problem. The histogram tool of FIG. **29** can provide guidance on the extent of a problem facing a designer.

A histogram tool provides a mechanism for displaying the distribution of a particular numerical characteristic of the circuit and allowing a user to see a list of objects having that characteristic. The example in FIG. **29** displays timing analysis for the AMD **2910**. The database uses a timing analyzer and a targeted clock period to compute the slack times on every net. The circuit nodes with similar slack times are grouped into bins, and counted. Histogram-list window **2900** contains two sub-windows: the histogram window **2920** and the list window **2910**. The histogram window **2920** contains bars **2930** through **2937** with one bar per bin. The height of the bars indicates the number of nets that fall into that bin. If the user selects one of the bars, the list window **2910** shows the list of names that identify the nets that are in that bin. Individual items in the list display can be selected, as indicated by selected item **2915**. This allows the designer to use other display tools to gain more information about the selected item.

Text Display

FIG. **30** shows the text display of HDL source code that annotates that source code with additional information. Text display window **3000** contains three smaller windows. Text window **3010** contains the source text. Select window **3040** shows circuit information related to text that has been selected. Cursor window **3030** shows circuit information related to text that is under the cursor. Column report **3060** shows circuit information associated with each line of the text. Selecting text can be done with the usual window based text selection mechanisms. For example, the designer could move a cursor to the relevant portion of the screen and push a button. The text window **3010** constructs a text box **3020** around the object that the cursor is pointing at. One fast method of determining the limits of cursor text box **3020** would be to use the parse tree representation described in co-pending application by Gregory entitled "Method and Apparatus for Context Sensitive Displays", filed on Jun. 3, 1994 with express mail certificate TB596163183US.

Here, cursor window **3030** is showing an estimate of the circuit area associated with the object under the cursor. The phrase "gtech Area=6" indicates that the implementation of the comparison function performed by condition "PSH_PTR>STK_LOW" indicated by cursor text box **3020** requires 6 area units in the generic technology domain in the database. Cursor window **3030** could display other characteristics associated with the text pointed to by the cursor.

Select window **3040** shows information associated with selected text. Here, the size and font of the selected text **1050** is changed. One fast method of determining the limits of selected text **1050** would be to use the parse tree representation described in co-pending application by Gregory entitled "Method and Apparatus for Context Sensitive Displays", filed on Jun. 3, 1994 with express mail certificate TB596163183US. In this example, the select window **3040** shows detailed information about the HDL construct MEM [PSH_PTR] **3050**. Statement **3080** shows the type of HDL construct that the selected object is—in this case, the construct is an array index. Statement **3083** shows the estimated area of the construct in the G-Tech domain **1510**, here 530 area units. Statement **3084** shows the length of the longest path in levels of logic from a register to the gates that implement the construct, here 18 gates. Statement **3081** shows the parse tree node number. A detailed list **3082** shows the netlist components in the G-Tech domain implementing the construct **3050**. For each component in this list, the following information is displayed: the component's netlist instance name **3087**, the type of netlist component **3088** (reference name), its contribution to the total area estimate **3089**, and the class of netlist component **3090** e.g. cell, pin, net, or port. Other information could be displayed at the designer's option.

Column report **3060** shows information associated with each line. Here, the column report is showing the area associated with the HDL constructs on each line.

Virtual Schematic Display

Part of the circuit debugging process involves tracing the drivers and driven or, inputs and outputs of specific circuitry. The virtual schematic display shown in FIG. **31** provides the designer with the ability to find the HDL source that provides inputs to and takes outputs from a particular point in the HDL source. The virtual schematic display has a virtual schematic window **3100** which has three window regions: an input region **3110**, a current region **3120**, and an output region **3130**. The designer uses the cursor to indicate selected text **3150** in the current region. This selected text **3150** corresponds to a circuit object **3151** (not shown) in the database. Circuit object **3151** has inputs and outputs. The database then links the input region **3110** to those portions of the text of the synthesis source that show where the inputs of circuit object **3151** originated. Here, the input "DATA" comes from an input to the module MULTIPLEXOR as indicated by input argument **3145**. The database also links the output region **3130** to those portions of the text of the synthesis source that show where the outputs of circuit object **3151** go to. Here, as indicated by output argument **3155**, output "Z" goes to the output of module MULTIPLEXOR.

By clicking in the output region, the designer can trace the output further. FIG. **32** shows the changes that occur in the regions. The text of the output region now moves to the current region **3120**. The text of the output region changes to show synthesis source text corresponding to objects driven by output argument **3155**. Here, output argument **3155** drives another output at the next level module boundary, as shown by output argument **3156**. Input region **3110** changes to show all of the places in the source text that set or define the selected output argument **3155**. Here, there are five text sources for output argument **3155** as shown in windows **3271**, **3272**, **3273**, **3274**, and **3275**. The originally selected statement **3150** is shown again in window **3272**.

An additional input comes from the MULTIPLEXOR input argument SEL as shown in window **3271**. Z is also takes on values at different points in a case statement as shown in windows **3272**, **3273**, **3274**, and **3275**.

FIG. **33** shows the results of pursuing the output argument **3156**. Here, the high level module definition appears in the current region **3120** while the input region **3110** displays the module interface shown in the current region of FIG. **32**.

What is claimed is:

1. A method, comprising:

translating a synthesis source text file to an initial circuit including one or more parts connected together with nets;

identifying each part of said initial circuit with the portion of said synthesis source text file that created said part of said initial circuit;

optimize said initial circuit to produce a final circuit containing one or more parts connected together with nets;

analyzing said final circuit to determine characteristics associated with said final circuit's parts and with said final circuit's nets;

identifying said final circuit's part's that correspond directly with said initial circuit's initial parts;

identifying said final circuit's nets that correspond directly with said initial circuit's nets; and

displaying said characteristics associated with those said final circuit's nets and parts that correspond directly with said initial circuit's nets and parts near said portions of said synthesis source text file that created said corresponding initial circuit parts and nets.

2. The method of claim **1** wherein said source text file includes directives indicating the location of probes and wherein said translating step include generating initial circuit parts or initial circuit nets that will correspond directly with said final circuit parts or final circuit nets.

* * * * *

U.S. Patent No. 6,240,376

US006240376B1

(12) **United States Patent**
Raynaud et al.

(10) **Patent No.: US 6,240,376 B1**
(45) **Date of Patent: May 29, 2001**

(54) METHOD AND APPARATUS FOR GATE-LEVEL SIMULATION OF SYNTHESIZED REGISTER TRANSFER LEVEL DESIGNS WITH SOURCE-LEVEL DEBUGGING

(75) Inventors: **Alain Raynaud**, Paris; **Luc M. Burgun**, Creteil, both of (FR)

(73) Assignee: **Mentor Graphics Corporation**, Wilsonville, OR (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/127,584**

(22) Filed: **Jul. 31, 1998**

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/122,493, filed on Jul. 24, 1998.

(51) **Int. Cl.**7 .... **G06F 17/50**
(52) **U.S. Cl.** **703/15**; 703/14; 714/741
(58) **Field of Search** 395/500.35, 500.36, 395/500.37; 714/724, 734; 703/14, 15, 16; 716/4, 724, 734, 741

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,220,512		6/1993	Watkins et al.	364/489
5,253,255	*	10/1993	Carbine	714/734

(List continued on next page.)

OTHER PUBLICATIONS

Chen et al., "A Source–Level Dynamic Analysis Methodology and Tool for High–LevelSynthesis", Proceedings of the Tenth International Symposium on System Synthesis, 1997, pp. 134–140, Sep. 1997.*

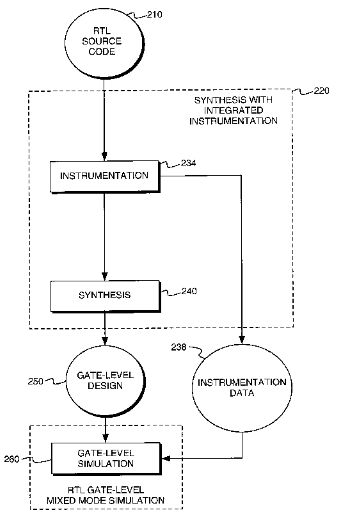
(List continued on next page.)

Primary Examiner—Kevin J. Teska
Assistant Examiner—Douglas W. Sergent
(74) *Attorney, Agent, or Firm*—Columbia IP Law Group, LLC

(57) ABSTRACT

Methods of instrumenting synthesizable source code to enable debugging support akin to high-level language programming environments for gate-level simulation are provided. One method of facilitating gate level simulation includes generating cross-reference instrumentation data including instrumentation logic indicative of an execution status of at least one synthesizable register transfer level (RTL) source code statement. A gate-level netlist is synthesized from the source code. Evaluation of the instrumentation logic during simulation of the gate-level netlist facilitates simulation by indicating the execution status of a corresponding source code statement. One method results in a modified gatelevel netlist to generate instrumentation signals corresponding to synthesizable statements within the source code. This may be accomplished by modifying the source code or by generating the modified gate-level netlist as if the source code was modified during synthesis. Alternatively, cross-reference instrumentation data including instrumentation logic can be generated without modifying the gate-level design. The instrumentation logic indicates the execution status of a corresponding cross-referenced synthesizable statement. An execution count of a cross-referenced synthesizable statement can be incremented when the corresponding instrumentation signals indicates the statement is active to determine source code coverage. Source code statements can be highlighted when active for visually tracing execution paths. For breakpoint simulation, a breakpoint can be set at a selected source code statement. The corresponding instrumentation logic from the cross-reference instrumentation data is implemented as a simulation breakpoint. The simulation is halted at a simulation cycle where the values of the instrumentation signals indicate that the source code statement is active.

33 Claims, 22 Drawing Sheets



U.S. PATENT DOCUMENTS

5,325,309		6/1994	Halaviati et al.	364/488
5,423,023		6/1995	Batch et al.	395/500
5,461,576		10/1995	Tsay et al.	364/490
5,513,123	*	4/1996	Dey et al.	716/4
5,519,627		5/1996	Mahamood et al.	364/488
5,541,849		7/1996	Rostoker et al.	364/489
5,544,067		8/1996	Rostoker et al.	364/489
5,553,002		9/1996	Dangelo et al.	364/489
5,555,201		9/1996	Dangelo et al.	364/489
5,568,396		10/1996	Bamji et al.	364/491
5,598,344		1/1997	Dangelo et al.	364/489
5,615,356		3/1997	King et al.	395/500
5,623,418		4/1997	Rostoker et al.	364/489
5,632,032	*	5/1997	Ault et al.	709/100
5,727,187		3/1998	Lemche et al.	395/500
5,758,123		5/1998	Sano et al.	395/500
5,768,145		6/1998	Roethig	364/488
5,801,958		9/1998	Dangelo et al.	364/489
5,835,380		11/1998	Roethig	364/488
5,841,663		11/1998	Sharma et al.	364/490
5,870,308		2/1999	Dangelo et al.	364/489
5,870,585	*	2/1999	Stapleton	703/15
5,880,971	*	3/1999	Dangelo et al.	703/16
5,920,711		7/1999	Seawright et al.	395/500
5,937,190		8/1999	Gregory et al.	395/704
5,960,191		9/1999	Sample et al.	395/500.49
5,991,533		11/1999	Sano et al.	395/500.49
6,006,022		12/1999	Rhim et al.	395/500.02
6,009,256		12/1999	Tseng et al.	395/500.34

OTHER PUBLICATIONS

Kucukcakar et al., "Matisse: An Architectural Design Tool for Commodity IC's", IEEE Design & Test of Computers, vol. 15, Issue 2, pp. 22–33, Jun. 1998.*

Koch et al. "Debugging of Beharioral VHDL Specifications by Source Level Emulation", Proceedings of the European Design Automation Conference, pp. 256–261, Sep. 1995.*

Fang et al., "A Real–Time RTL Engineering–Change Method Supporting On–Line Debugging for Logic–Emulation Applications", Proceedings of the 34th Design Automation Conference, pp. 101–106, Jun. 1997.*

Howe, H., "Pre– and Postsynthesis Mismatches", IEEE International Conf. on Verilog HDL 1997, pp. 24–31, Apr. 1997.*

Postula et al., "A Comparison of High Level Synthesis and Register Transfer Design Techniques for Custom Computing Machines", Proc. of the 31st Hawaii Inter. Conf. on System Sciences, vol. 7, pp. 207–214, Jan. 1998.*

Orailoglu, A., "Microarchitectural Sythesis for Rapid BIST Testing", IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems, vol. 16, Issue 6, pp. 573–586, Jun. 1997.*

* cited by examiner

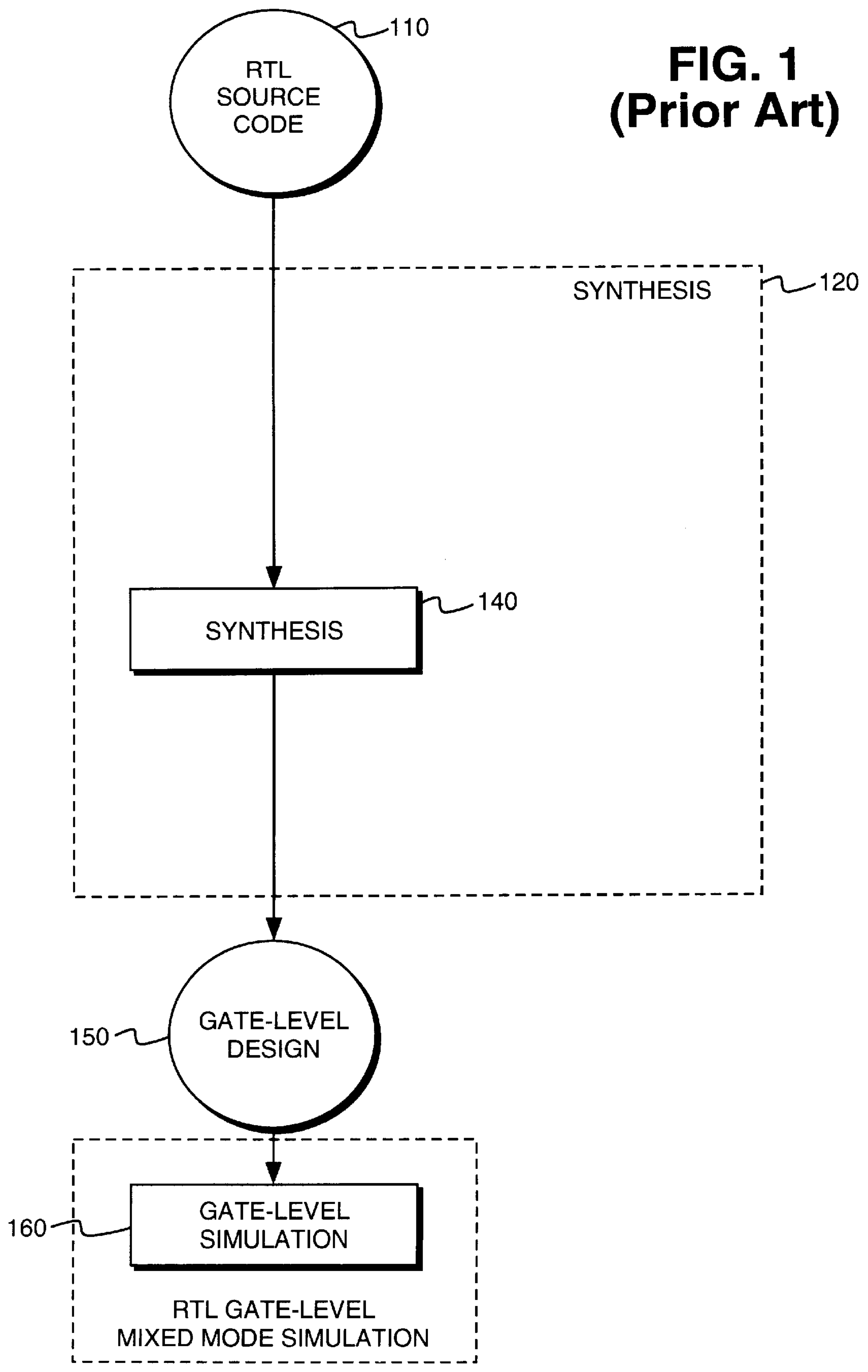
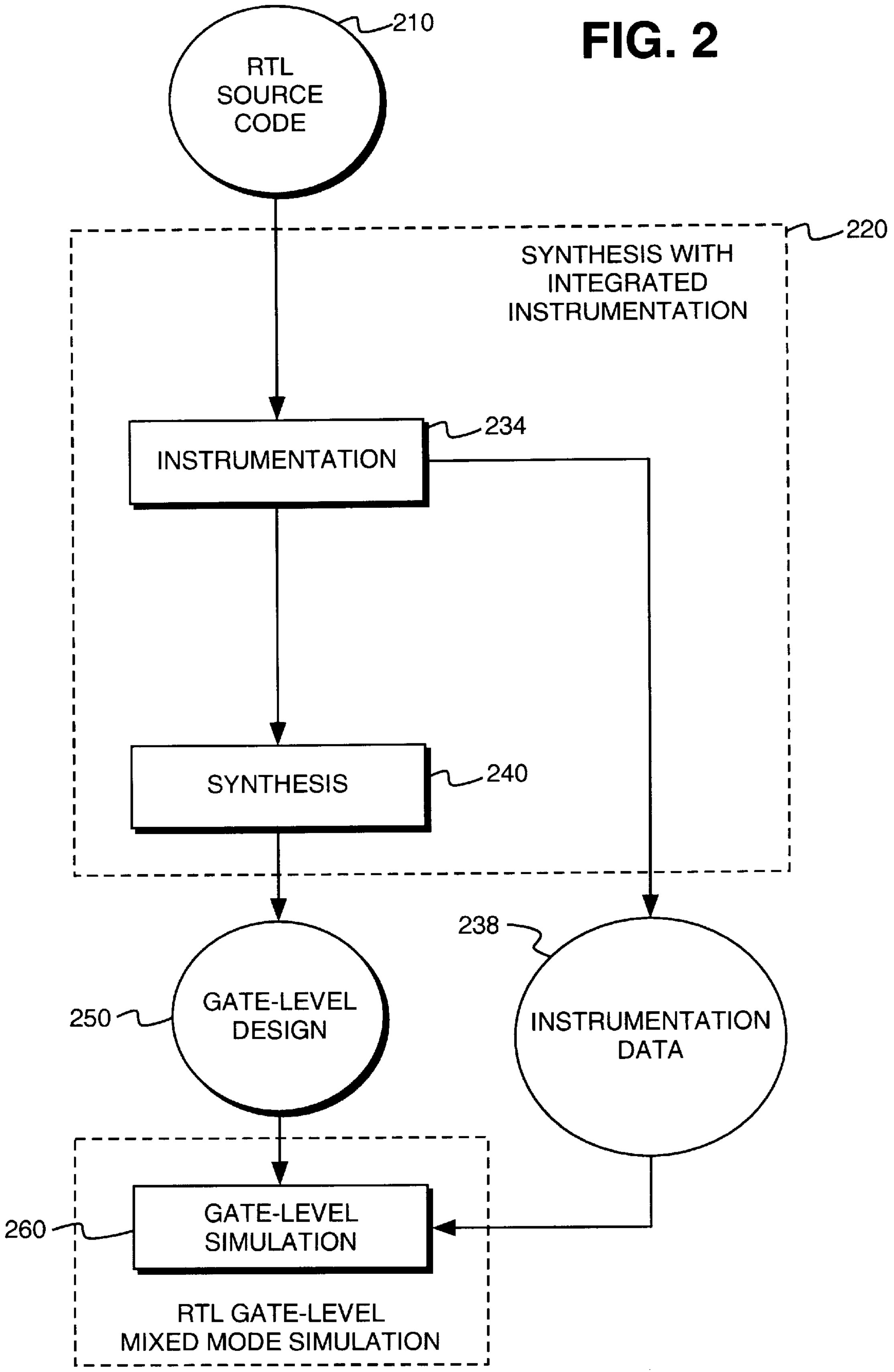


FIG. 1 (Prior Art)

FIG. 2



RTL SOURCE CODE
210
220
SYNTHESIS WITH INTEGRATED INSTRUMENTATION
INSTRUMENTATION
234
SYNTHESIS
240
238
GATE-LEVEL DESIGN
250
INSTRUMENTATION DATA
GATE-LEVEL SIMULATION
260
RTL GATE-LEVEL MIXED MODE SIMULATION

FIG. 3

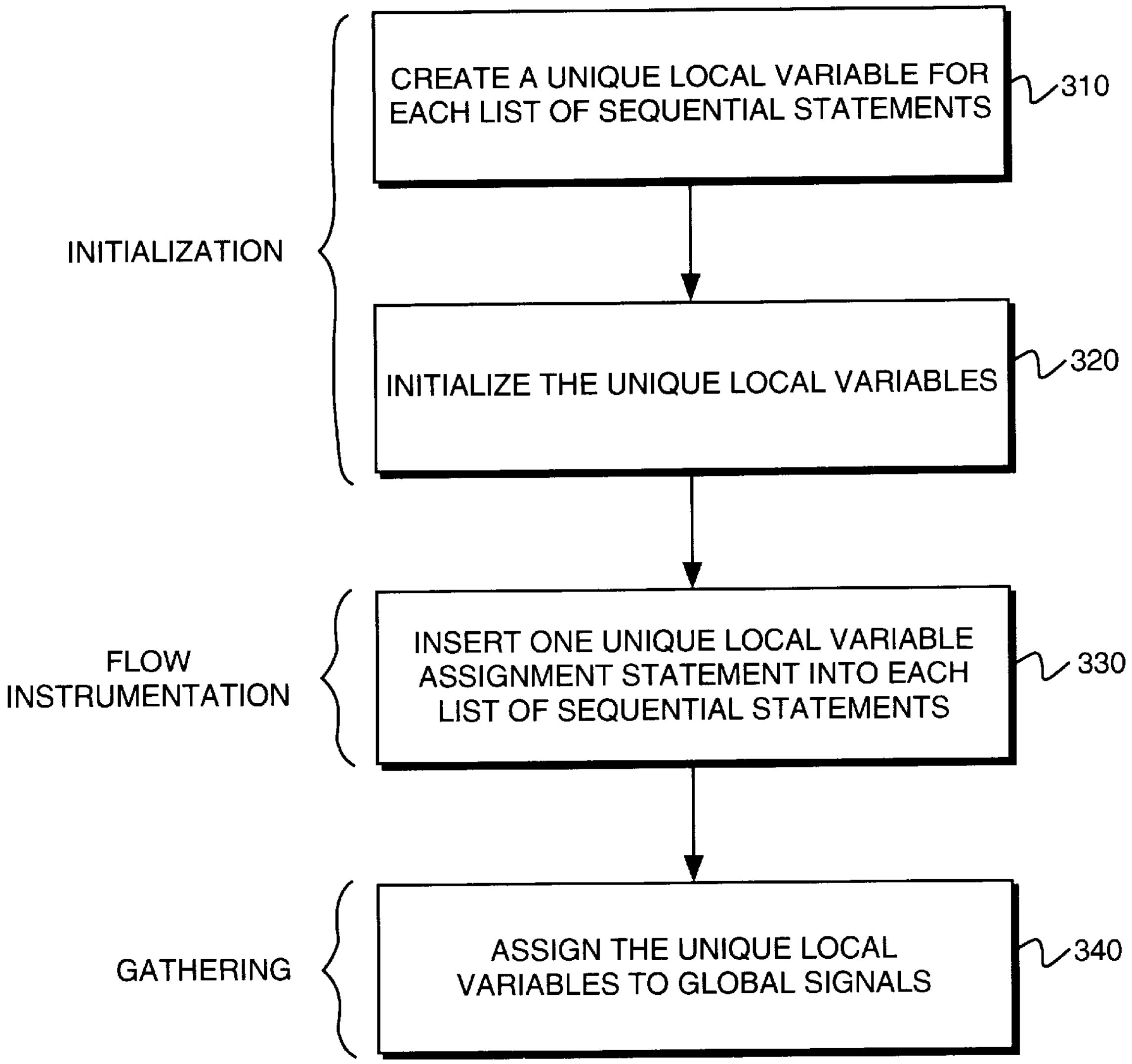

CREATE A UNIQUE LOCAL VARIABLE FOR EACH LIST OF SEQUENTIAL STATEMENTS
310
INITIALIZATION
INITIALIZE THE UNIQUE LOCAL VARIABLES
320
FLOW INSTRUMENTATION
INSERT ONE UNIQUE LOCAL VARIABLE ASSIGNMENT STATEMENT INTO EACH LIST OF SEQUENTIAL STATEMENTS
330
GATHERING
ASSIGN THE UNIQUE LOCAL VARIABLES TO GLOBAL SIGNALS
340

FIG. 4

400

```
          ENTITY ALOOP IS

          PORT(
                A : IN BIT_ VECTOR ( 0 TO 1 ) ;
                RESET : IN BOOLEAN;
                STATUS : OUT BOOLEAN ) ;

          END ENTITY ALOOP ;


          ARCHITECTURE RTL OF ALOOP IS

          BEGIN

          PROCESS ( A, RESET )

          VARIABLE ZEROS, ONES : INTEGER ;

          BEGIN
410 ───►        IF ( RESET )                              -- STATEMENT # 1
                THEN
420 ───►              STATUS <= 0 ;                       -- STATEMENT # 2
                ELSE
430 ───►              ZEROS := 0;                         -- STATEMENT # 3
440 ───►              ONES := 0;                          -- STATEMENT # 4
450 ───►              FOR I IN 0 TO 1 LOOP                -- STATEMENT # 5
460 ───►                    IF A ( I ) = '0'              -- STATEMENT # 6
                            THEN
470 ───►                          ZEROS := ZEROS + 1 ;    -- STATEMENT # 7
                            ELSE
480 ───►                          ONES := ONES + 1 ;      -- STATEMENT # 8
                            END IF ;

                      END LOOP ;

490 ───►              STATUS <= ( ZEROS > ONES ) ;        -- STATEMENT # 9
                END IF ;

          END PROCESS ;

          END ARCHITECTURE ;
```

FIG. 5

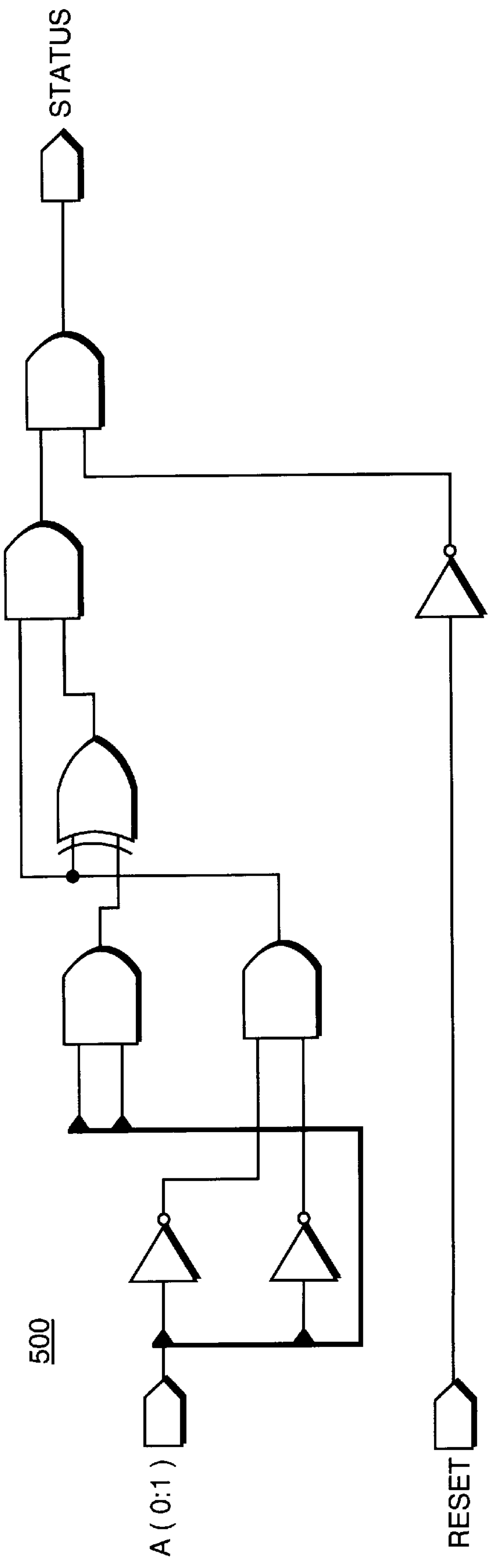
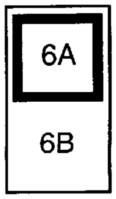


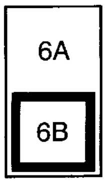
FIG. 6A



600

```
ENTITY ALOOP IS
PORT(
A : IN BIT_VECTOR (0 TO 1) ;
RESET : IN BOOLEAN ;
STATUS : OUT BOOLEAN ;
SIG_ TRACE1, SIG_ TRACE2, SIG_ TRACE3, SIG_ TRACE4, SIG_TRACE5,  } 610
SIG_TRACE6 : OUT BIT
);
END ENTITY ALOOP ;

ARCHITECTURE RTL OF ALOOP IS
BEGIN
PROCESS (A, RESET)
VARIABLE TRACE1, TRACE2, TRACE3, TRACE4, TRACE5, TRACE6 : BIT ;  } 612
VARIABLE ZEROS, ONES : INTEGER ;

BEGIN
TRACE1 :='0' ; TRACE2 :='0' ;
TRACE3 :='0' ; TRACE4 :='0' ;  } 620
TRACE5 :='0' ; TRACE6 :='0' ;
```

FIG. 6B



600

```
630→TRACE1 := '1';                            -- INSTRUMENT STATEMENT #1
    IF (RESET)                                -- STATEMENT #1
    THEN
632────→TRACE2 := '1';                        -- INSTRUMENT STATEMENT #2
        STATUS <= FALSE ;                     -- STATEMENT #2
    ELSE
634────→TRACE3 := '1';                        -- INSTRUMENT STATEMENT #3, #4, #5, #9
        ZEROS := 0 ;                          -- STATEMENT #3
        ONES := 0 ;                           -- STATEMENT #4
        FOR I IN 0 TO 1 LOOP                  -- STATEMENT #5
636────────→TRACE4 := '1';                    -- INSTRUMENT STATEMENT #6
            IF A (I) ='0' ;                   -- STATEMENT #6
            THEN
638────────────→TRACE5 := '1';                -- INSTRUMENT STATEMENT #7
                ZEROS := ZEROS +1 ;           -- STATEMENT #7
            ELSE
640────────────→TRACE6 := '1';                -- INSTRUMENT STATEMENT #8
                ONES := ONES +1;              -- STATEMENT  #8
            END IF;
        END LOOP;

642─────→STATUS <= (ZEROS > ONES) ;           -- STATEMENT #9

    END IF ;
    SIG_ TRACE1 <= TRACE1 ; SIG_ TRACE2 <= TRACE2 ; \
    SIG_ TRACE3 <= TRACE3 ; SIG_ TRACE4 <= TRACE4 ;  }~ 650
    SIG_ TRACE5 <= TRACE5 ; SIG_ TRACE6 <= TRACE6 ; /
    END PROCESS ;

    END ARCHITECTURE ;
```

FIG. 7

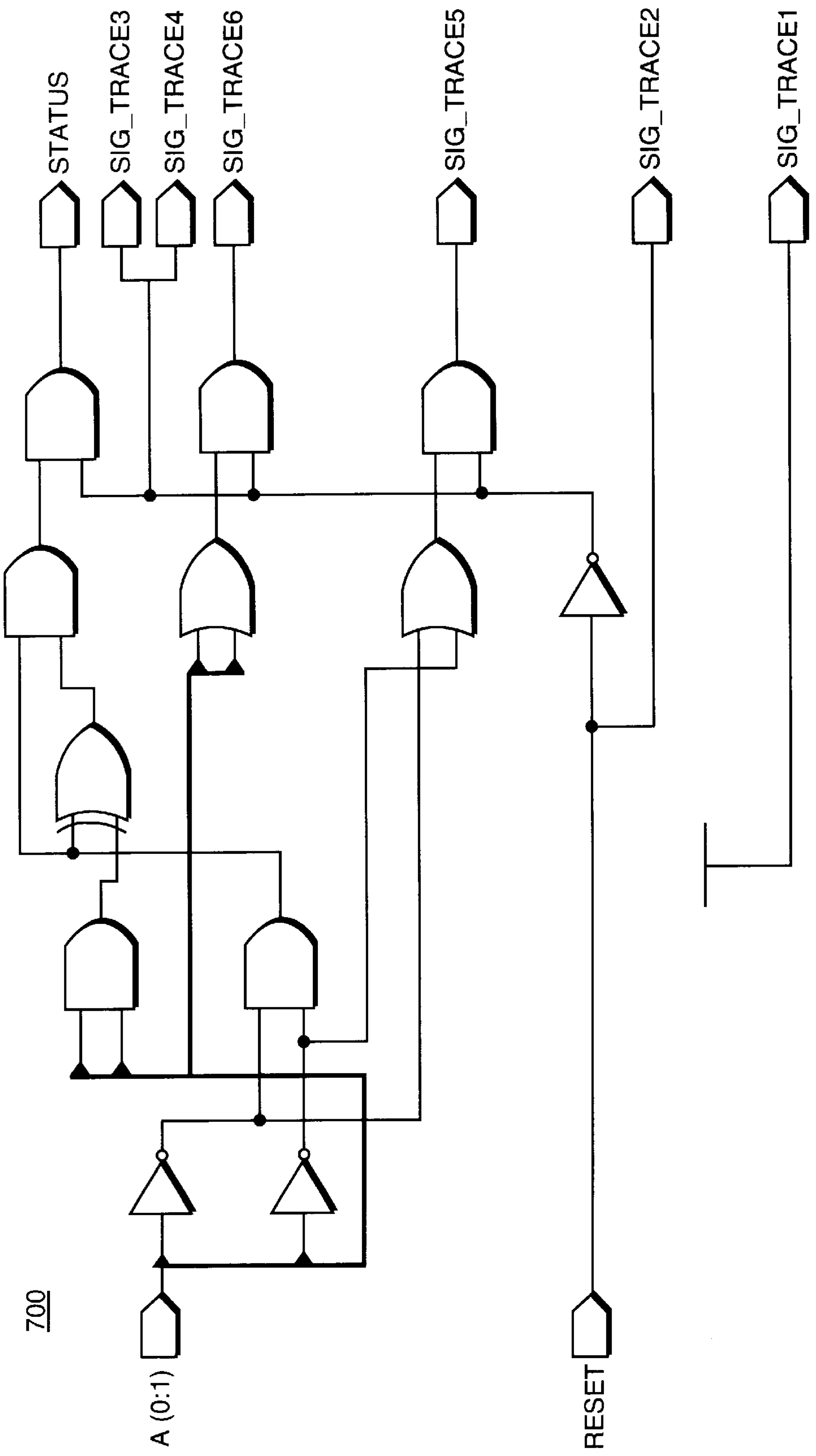

STATUS
SIG_TRACE3
SIG_TRACE4
SIG_TRACE6
SIG_TRACE5
SIG_TRACE2
SIG_TRACE1
700
A (0:1)
RESET

FIG. 8

800

```
MODULE SAMPLE (RESET, D, CLK, Q) ;

INPUT RESET ;
INPUT D ;
INPUT CLK ;
REG Q ;
OUTPUT Q ;


ALWAYS @ (CLK OR RESET OR D)
BEGIN
            IF (RESET==1)
                  Q <= 0 ;
            ELSE
                  IF (CLK==1)
                      Q <= D ;

END


ENDMODULE
```

FIG. 9

900

```
MODULE SAMPLE
(RESET, D, CLK, Q, SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_ TRACE4 ) ;

INPUT RESET ;
INPUT D ;
INPUT CLK ;
REG Q ;
OUTPUT Q ;

REG SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_TRACE4 ;
OUTPUT SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_TRACE4 ;

INTEGER TRACE1, TRACE2, TRACE3, TRACE4 ;

ALWAYS @ (CLK OR RESET OR D)
BEGIN
        TRACE1 = 0 ; TRACE2 = 0 ; TRACE3 = 0 ; TRACE4 = 0 ;
        TRACE1 = 1 ;
        IF (RESET==1)
        BEGIN
                TRACE2 = 1 ;
                Q <= 0 ;
        END
        ELSE
        BEGIN
                TRACE3 = 1 ;
                IF (CLK== 1)
                BEGIN
                        TRACE4 = 1 ;
                           Q <= D ;
                END
        END

SIG_TRACE1 = TRACE1 ;
SIG_TRACE2 = TRACE2 ;
SIG_TRACE3 = TRACE3 ;
SIG_TRACE4 = TRACE4 ;

END

ENDMODULE
```

FIG. 10

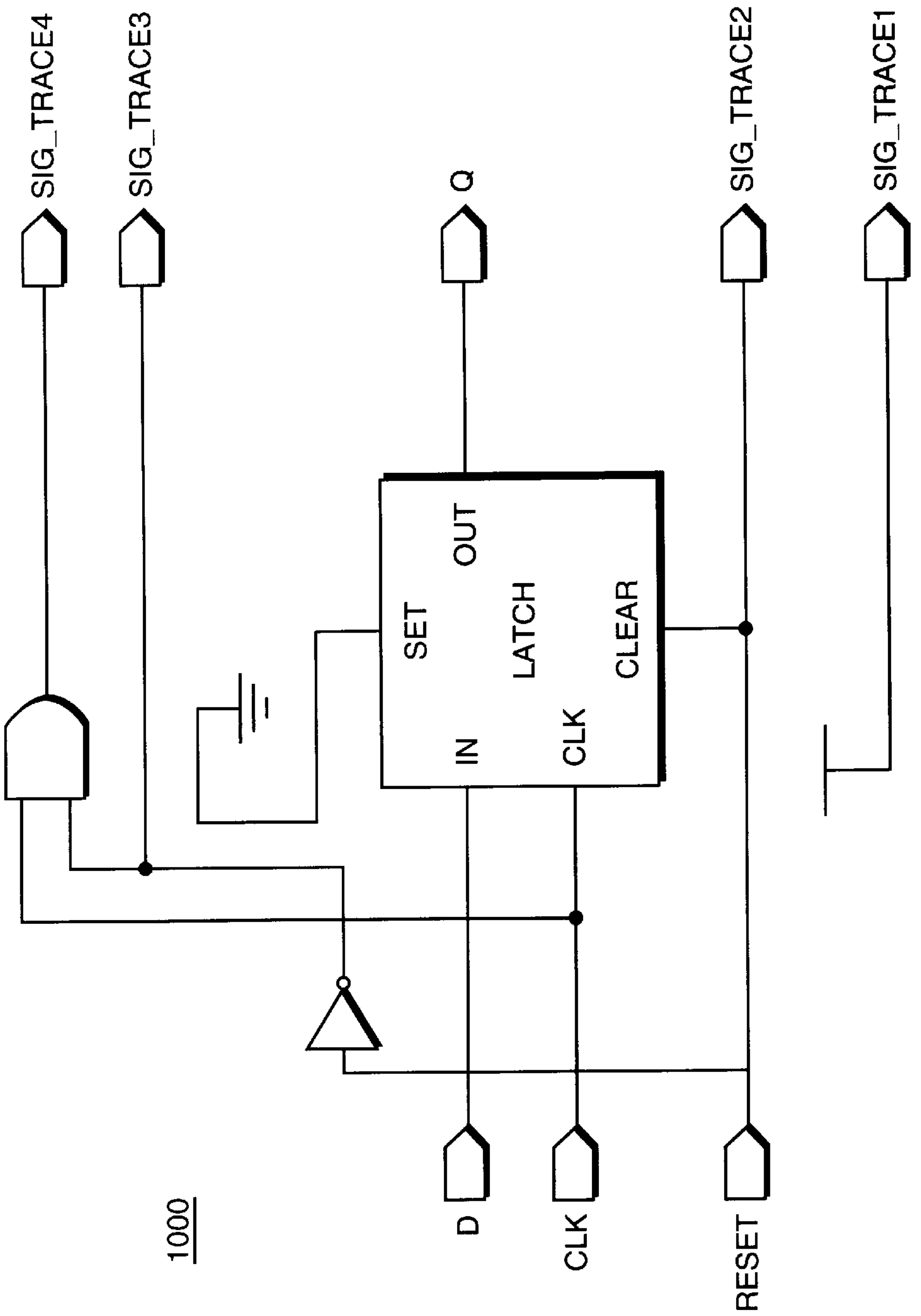

1000
SIG_TRACE4
SIG_TRACE3
Q
SIG_TRACE2
SIG_TRACE1
SET
OUT
LATCH
CLEAR
IN
CLK
D
CLK
RESET

FIG. 11

1100

```
PROCESS (CLK, D, RESET)
BEGIN

        IF (RESET = '1') THEN
                Q <= '0' ;

        ELSIF (CLK'EVENT AND CLK = '1') THEN
                Q <= D ;

        END IF;

END PROCESS
```

FIG. 12

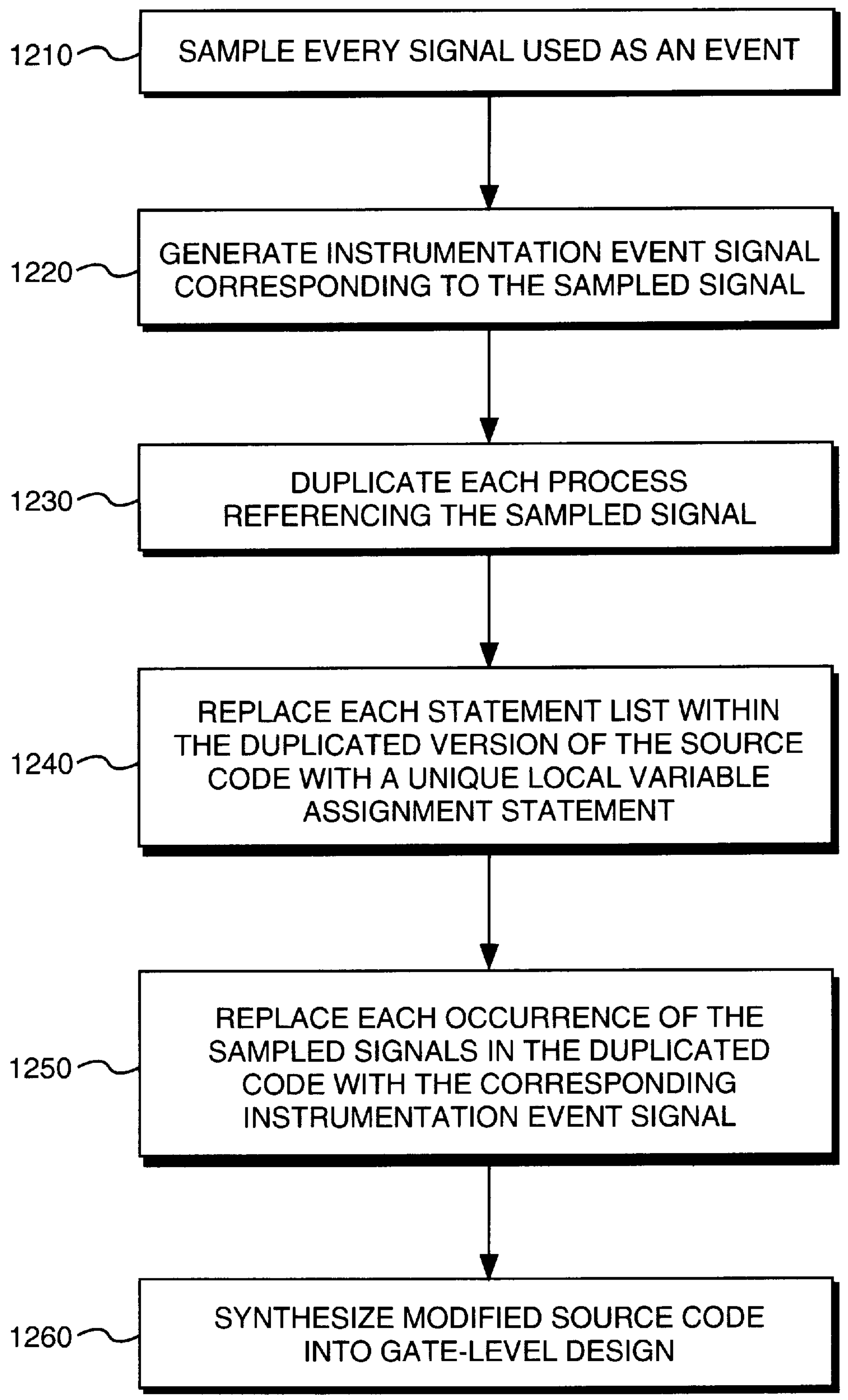

1210
SAMPLE EVERY SIGNAL USED AS AN EVENT
1220
GENERATE INSTRUMENTATION EVENT SIGNAL CORRESPONDING TO THE SAMPLED SIGNAL
1230
DUPLICATE EACH PROCESS REFERENCING THE SAMPLED SIGNAL
1240
REPLACE EACH STATEMENT LIST WITHIN THE DUPLICATED VERSION OF THE SOURCE CODE WITH A UNIQUE LOCAL VARIABLE ASSIGNMENT STATEMENT
1250
REPLACE EACH OCCURRENCE OF THE SAMPLED SIGNALS IN THE DUPLICATED CODE WITH THE CORRESPONDING INSTRUMENTATION EVENT SIGNAL
1260
SYNTHESIZE MODIFIED SOURCE CODE INTO GATE-LEVEL DESIGN

FIG. 13

1300

```
PROCESS (FAST_CLK)
BEGIN
        IF (FAST_CLK'EVENT AND FAST_CLK = '1')
        THEN
                SAMPLED_CLK <= CLK ;
        END IF
END PROCESS ;

CLK_EVENT <= SAMPLED_CLK / = CLK ;
CLK_STABLE <= SAMPLED_CLK = CLK ;
CLK_LASTVALUE <= SAMPLED_CLK ;
```

1310

```
PROCESS (CLK, D, RESET, CLK _ EVENT)
        VARIABLE TRACE1, TRACE2 : BIT ;
BEGIN
        TRACE1 := '0' ; TRACE2 := '0' ;
        IF (RESET = '1') THEN
                TRACE1 := '1' ;
        ELSIF (CLK_EVENT AND CLK = '1') THEN
                TRACE2 := '1' ;
        END IF;
SIG_TRACE1 <= TRACE1 ;  SIG_TRACE2 <= TRACE2 ;
END PROCESS ;
```

1320

```
PROCESS (CLK, D, RESET)
BEGIN
        IF (RESET = '1') THEN
                Q <= '0' ;
        ELSIF (CLK'EVENT AND CLK = '1') THEN
                Q <= D ;
        END IF ;
END PROCESS
```

1330

FIG. 14

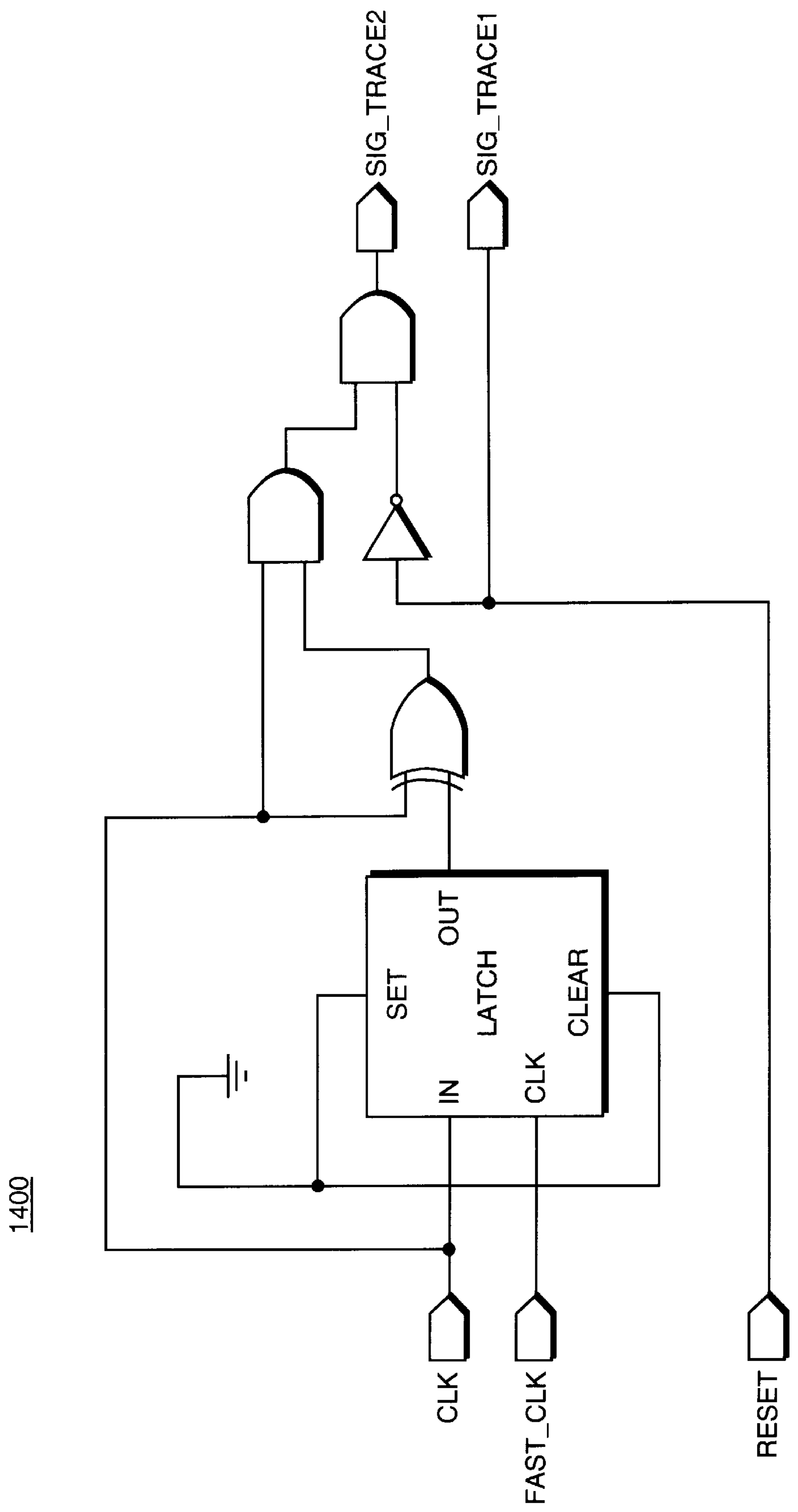

1400
SIG_TRACE2
SIG_TRACE1
SET
OUT
LATCH
IN
CLK
CLEAR
CLK
FAST_CLK
RESET

FIG. 15

1500

```
ALWAYS @ (POSEDGE CLK OR NEGEDGE RESET)
BEGIN
        IF (RESET == 0)
                  Q <= 0 ;
        ELSE
                  Q <= D ;
END
```

FIG. 16

1600

```
ALWAYS @ (POSEDGE FAST_CLK)
BEGIN
        SAMPLED_CLK <= CLK
        SAMPLED_RESET <= RESET ;
END
ASSIGN CLK_EDGE = SAMPLED_CLK ^ CLK ;
ASSIGN RESET_ EDGE = SAMPLED_ RESET ^ RESET;

INTEGER TRACE1, TRACE2;
REG [1:0] SIG_ TRACE ;
ALWAYS @ (CLK_EDGE OR RESET_EDGE OR CLK OR RESET)
BEGIN
       TRACE1 = 0 ; TRACE2 = 0 ;
       IF ((CLK_EDGE == 1) && (CLK == 1)II(RESET_EDGE == 1) && (RESET == 0))
                IF (RESET == 0)
                         TRACE1 =1;
                ELSE
                         TRACE2 =1;
                SIG_TRACE[0] = TRACE1 ;
                SIG_TRACE[1] = TRACE2 ;
END

ALWAYS @ (POSEDGE CLK OR NEGEDGE RESET)
BEGIN


       IF (RESET == 0)
            Q <=0 ;
        ELSE
            Q <= D ;
END
```

FIG. 17

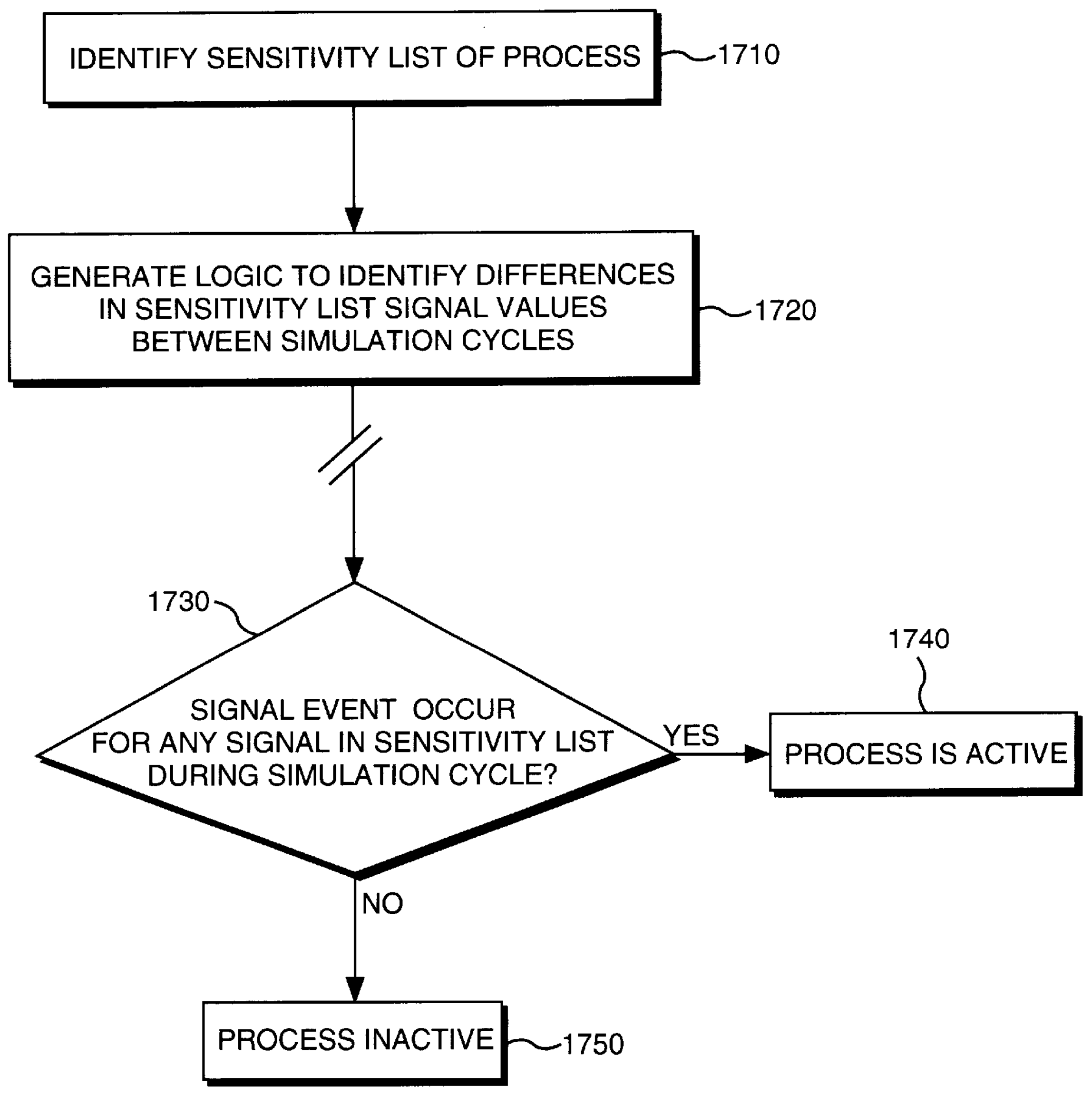

IDENTIFY SENSITIVITY LIST OF PROCESS
1710
GENERATE LOGIC TO IDENTIFY DIFFERENCES IN SENSITIVITY LIST SIGNAL VALUES BETWEEN SIMULATION CYCLES
1720
1730
SIGNAL EVENT OCCUR FOR ANY SIGNAL IN SENSITIVITY LIST DURING SIMULATION CYCLE?
YES
1740
PROCESS IS ACTIVE
NO
PROCESS INACTIVE
1750

FIG. 18

```
P1: PROCESS (A,B,C)

PROCESS (FAST_CLK)
 BEGIN
          IF (FAST_ CLK'EVENT AND FAST_ CLK='1')
          THEN
                  SAMPLED_A <=A ;
                  SAMPLED_B <=B ;                         1810
                  SAMPLED_C <=C ;
          END IF
          END PROCESS;


P1_ ACTIVE <= (SAMPLED_A /= A)
       OR (SAMPLED_B  /= B)                               1820
       OR (SAMPLED_C  /= C);
```

FIG. 19

```
      CASE OPCODE IS
          WHEN "00" => TRACE1 := 1;
1910                   STATE := 1;
          WHEN "01" => TRACE2 := 1;
                       STATE:= 2 ;
          WHEN "10" => TRACE3 := 1;
                       STATE := 2 ;
          WHEN "11" => TRACE4 := 1;
                       STATE := 1 ;
      END CASE ;
```

FIG. 20

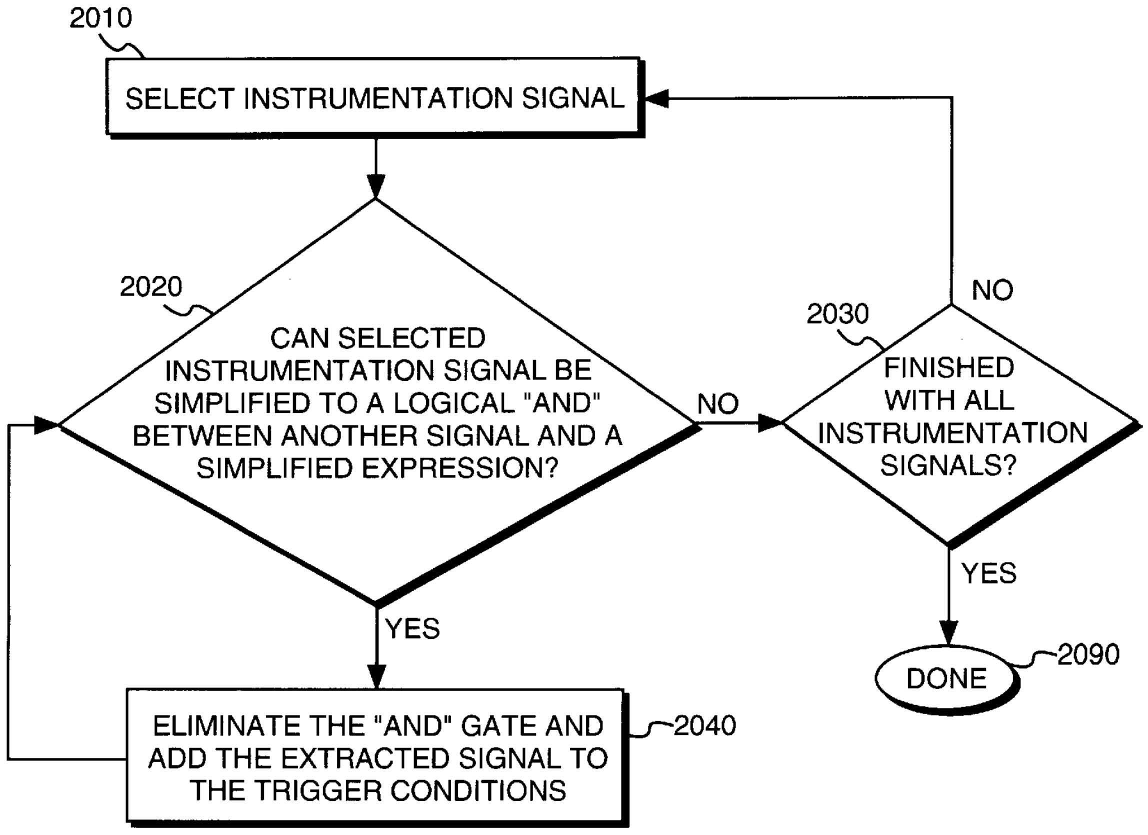

2010
SELECT INSTRUMENTATION SIGNAL
2020
CAN SELECTED INSTRUMENTATION SIGNAL BE SIMPLIFIED TO A LOGICAL "AND" BETWEEN ANOTHER SIGNAL AND A SIMPLIFIED EXPRESSION?
NO
YES
2030
FINISHED WITH ALL INSTRUMENTATION SIGNALS?
NO
YES
ELIMINATE THE "AND" GATE AND ADD THE EXTRACTED SIGNAL TO THE TRIGGER CONDITIONS
2040
DONE
2090

FIG. 21

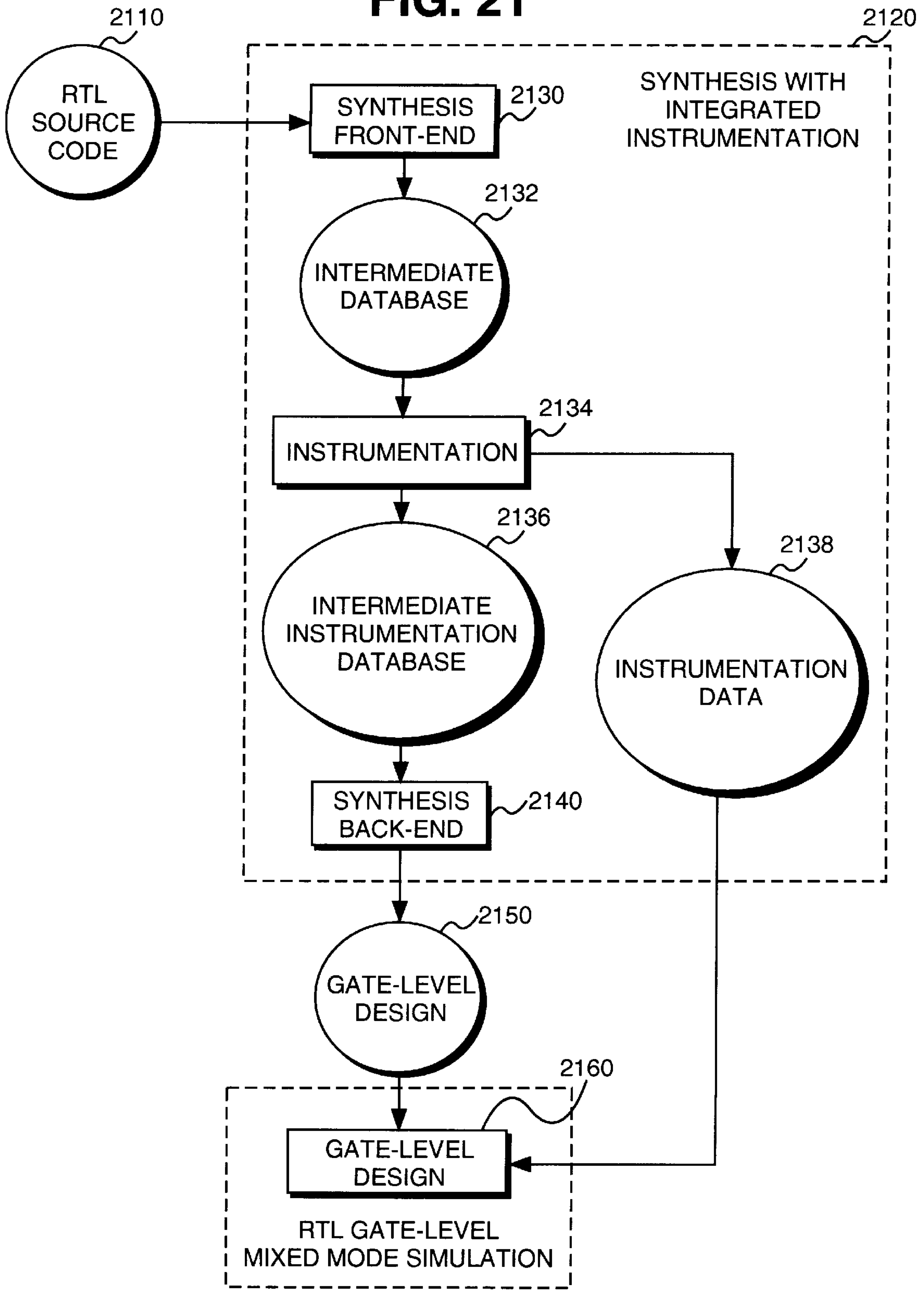
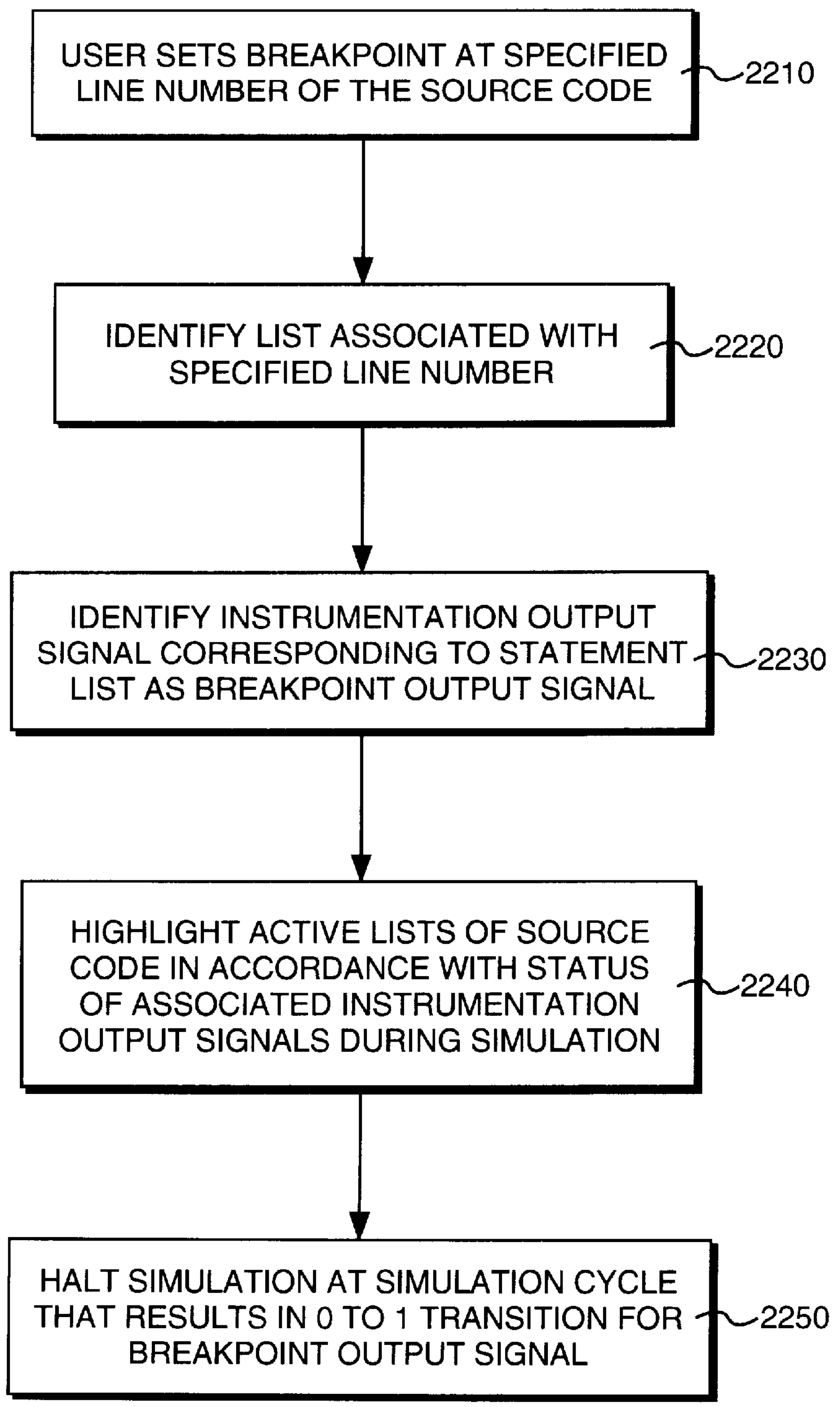

2110
RTL SOURCE CODE
2120
SYNTHESIS WITH INTEGRATED INSTRUMENTATION
SYNTHESIS FRONT-END
2130
2132
INTERMEDIATE DATABASE
2134
INSTRUMENTATION
2136
INTERMEDIATE INSTRUMENTATION DATABASE
2138
INSTRUMENTATION DATA
SYNTHESIS BACK-END
2140
2150
GATE-LEVEL DESIGN
2160
GATE-LEVEL DESIGN
RTL GATE-LEVEL MIXED MODE SIMULATION

FIG. 22


USER SETS BREAKPOINT AT SPECIFIED LINE NUMBER OF THE SOURCE CODE
2210
IDENTIFY LIST ASSOCIATED WITH SPECIFIED LINE NUMBER
2220
IDENTIFY INSTRUMENTATION OUTPUT SIGNAL CORRESPONDING TO STATEMENT LIST AS BREAKPOINT OUTPUT SIGNAL
2230
HIGHLIGHT ACTIVE LISTS OF SOURCE CODE IN ACCORDANCE WITH STATUS OF ASSOCIATED INSTRUMENTATION OUTPUT SIGNALS DURING SIMULATION
2240
HALT SIMULATION AT SIMULATION CYCLE THAT RESULTS IN 0 TO 1 TRANSITION FOR BREAKPOINT OUTPUT SIGNAL
2250

METHOD AND APPARATUS FOR GATE-LEVEL SIMULATION OF SYNTHESIZED REGISTER TRANSFER LEVEL DESIGNS WITH SOURCE-LEVEL DEBUGGING

This is a continuation in part of application Ser. No. 09/122,493, filed Jul. 4, 1998.

FIELD OF THE INVENTION

This invention relates to the fields of simulation and prototyping hen designing integrated circuits. In particular, this invention is drawn to debugging synthesizable code at the register transfer level during gate-level simulation.

BACKGROUND OF THE INVENTION

Integrated circuit designers have adopted the use of high-level hardware description languages due in part to the size and complexity of modem integrated circuits. One such description language is Very High Speed Integrated Circuit (VHSIC) Description Language, or VHDL. Further information regarding VHDL may be found in the IEEE Standard VHDL Language Reference Manual (IEEE 1076–1987, IEEE 1076–1993). Another such description language is Verilog. These high level description languages are typically generically referred to as hardware description languages (HDLs).

Synthesis is the process of generating a gate-level netlist from the high level description languages. Presently, synthesis tools recognize a subset of the high-level description language source code referred to as Register Transfer Level (RTL) source code. Further information regarding RTL source code may be found in the IEEE 1076.6/D1.10 Draft Standard for VHDL Register Transfer Level Synthesis (1997).

The RTL source code can be synthesized into a gate-level netlist. The gate-level netlist can be verified using gate-level simulation. The gate-level simulation can be performed using a software gate-level simulator. alternatively, the gate-level simulation may be performed by converting the ate-level netlist into a format suitable for programming an emulator, a hardware accelerator, or a rapid-prototyping system so that the digital circuit description can take an actual operating hardware form.

Debugging environments for high-level hardware description languages frequently include a number of functionalities for analyzing and verifying the design when performing simulation. For example, a designer can typically navigate the design hierarchy, view the RTh source code, and set breakpoints on a statement of RTL source code to stop the simulation. Statements are usually identified by their line number in the RTL source code. In addition, the debugging environment often supports viewing and tracing variables and signal values. The RTL simulation environment typically offers such RTL debugging functionalities.

RTL simulation is typically performed by using software RTL simulators which provide good flexibility. However, for complex designs, a very large number of test vectors may need to be applied in order to adequately verify the design. This can take a considerable amount of time using software RTL simulation as contrasted with hardware acceleration or emulation starting from a gate-level netlist representation (i.e., "gate-level hardware acceleration," or "gate level emulation"). Furthermore, it may be useful to perform in-situ verification, which consists of validating the design under test by connecting the emulator or hardware accelerator to the target system environment (where the design is to be inserted after the design is completed).

One disadvantage with gate-level simulation, however, is that most of the high-level information from the RTL source code is lost. Without the high-level information, many of the debugging functionalities are unavailable.

For example, the designer typically cannot set a breakpoint from the source code during gate-level simulation. Although signals can be analyzed during gate-level simulation, mapping signal values to particular source code lines can be difficult, if not impossible. If the source code is translated into a combinatorial logic netlist, for example, the designer cannot "step" through the source code to trace variable values. Instead, the designer is limited to analyzing the input vector and resulting output vector values. Although the signals at the inputs and outputs of the various gates may be traced or modified, these values are determined concurrently in a combinatorial network and thus such analysis is not readily mappable to the RTL source code.

A typical design flow will include creating a design at the RTL level, then synthesizing it into a gate-level netlist. Although simulation of this netlist can be performed at greater speeds using emulators or hardware accelerators, the ability to debug the design at the gate level is severely limited in comparison with software RTL simulation.

SUMMARY OF THE INVENTION

Methods of instrumenting synthesizable register transfer level (RTL) source code to enable debugging support akin to high-level language programming environments for gate-level simulation are provided.

One method of facilitating gate-level simulation includes the step of generating cross-reference instrumentation data including instrumentation logic indicative of the execution status of at least one synthesizable statement within the RTL source code. A gate-level netlist is synthesized from the RTL source code. Evaluation of the instrumentation logic during simulation of the gate-level netlist enables RTL debugging by indicating the execution status of the cross-referenced synthesizable statement in the RTh source code.

In one embodiment, the gate-level netlist is modified to provide instrumentation signals implementing the instrumentation logic and corresponding to synthesizable statements within the RTL source code. In various embodiments, this may be accomplished by modifying the RTL source code or by generating the modified gate-level netlist during synthesis as if the source code had been modified.

Alternatively, the gate-level netlist is not modified but the instrumentation signals implementing the instrumentation logic are contained in a cross-reference instrumentation database. In either case, the instrumentation signals indicate the execution status of the corresponding cross-referenced synthesizable statement. The instrumentation signals can be used to facilitate source code analysis, breakpoint debugging, and visual tracing of the source code execution path during gate-level simulation.

For example, a breakpoint can be set at a selected statement of the source code. A simulation breakpoint is set so that the simulation is halted at a simulation cycle where the value of the instrumentation signals indicate that the statement has become active .

With respect to visually tracing the source code during execution, the instrumentation logic is evaluated during gate-level simulation to determine a list of at least one active statement. The active statement is displayed as a highlighted statement.

With respect to source code analysis, cross-reference instrumentation data including the instrumentation signals

can be used to count the number of times a corresponding statement is executed in the source code. For example, an execution count of the cross-referenced synthesizable statement is incremented when evaluation of the corresponding instrumentation logic indicates that the cross-referenced synthesizable statement is active.

Other features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description that follows below.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

FIG. **1** illustrates the process of synthesizing RTL source code into a gate-level design.

FIG. **2** illustrates one embodiment of a modified process for generating a gate-level design.

FIG. **3** illustrates one embodiment of a method for instrumenting level-sensitive RTL source code.

FIG. **4** illustrates VHDL source code.

FIG. **5** illustrates the gate-level design synthesized from the RTL source code of FIG. **4**.

FIG. **6** illustrates the VHDL source code of FIG. **4** modified in accordance with the method of FIG. **3**.

FIG. **7** illustrates one embodiment of the gate-level logic synthesized from the modified RTL source code.

FIG. **8** illustrates sample Verilog source code before instrumentation.

FIG. **9** illustrates the Verilog source of FIG. **8** instrumented in accordance with the method of FIG. **3**.

FIG. **10** illustrates the gate-level logic synthesized from the instrumented Verilog source code of FIG. **9**.

FIG. **11** illustrates VHDL source code for a D flip-flop with asynchronous reset.

FIG. **12** illustrates one method of instrumenting event-sensitive RTL source code.

FIG. **13** illustrates the source code of FIG. **11** modified in accordance with the instrumentation process of FIG. **12**.

FIG. **14** illustrates the gate-level logic synthesized for the instrumented source code of FIG. **13**.

FIG. **15** illustrates Verilog source code for a D flip-flop with asynchronous reset.

FIG. **16** illustrates the Verilog source code of FIG. **15** after instrumentation in accordance with the method of FIG. **12**.

FIG. **17** illustrates a method of instrumenting process activation.

FIG. **18** illustrates source code modified in accordance with the method of FIG. **17**.

FIG. **19** illustrates an instrumented "case" statement.

FIG. **20** illustrates a process for decreasing the logic needed to instrument the source code.

FIG. **21** illustrates incorporating instrumentation within the synthesis process.

FIG. **22** illustrates a method of setting a breakpoint in RTL source code for use during gate-level simulation.

DETAILED DESCRIPTION

FIG. **1** illustrates a typical RTL source code synthesis process. HDL code including synthesizable RTL source code (**110**) serves as input to a synthesis process **120**. In one

embodiment, the RTL source code **110** is synthesized in step **140** to produce a gate-level design **150**. The gatelevel design can be used for gate-level simulation as illustrated in step **160**.

Typically the gate-level design comprises a hierarchical or flattened gate level netlist representing the circuit to be simulated. The various signals in a design are referred to as nets. A hierarchical netlist is made of a list of blocks, whereas a flattened netlist comprises only one block. A block contains components and a description of their interconnection using nets. Components can be reduced to combinatorial or sequential logic gates, or they may be hierarchical blocks of lower level.

For example, the component may be a primitive gate denoting a single combinatorial logic function (e.g., AND, NAND, NOR, OR, XOR, NXOR, etc.) or a single storage element such as a flip-flop or latch for sequential logic. One example of a set of primitive gates is found in the generic library GTECH available from Synopsys, Inc. of Mountain View, Calif.

Alternatively the component may be an application specific integrated circuit (ASIC) library cell which can be represented by a set of primitive gates. One example of an ASIC library is the LCA300K ASIC library developed by LSI Logic, Inc., Milpitas, Calif.

A component may also be a programmable primitive that represents a set of logic functions and storage. One example of a programmable primitive is the configurable logic block (CLB) as described in The Programmable Gate Array Handbook, Xilinx Inc., San Jose, 1993.

Another example of a component is a macro block denoting a complex logic function such as memories, counters, shifters, adders, multipliers, etc. Each of these can be further reduced to primitive gates forming combinatorial or sequential logic.

Three major categories of tools are available to the designer to simulate and test the design. Software RTL simulators (such as ModelSim™ from Model Technology, Inc.) typically offer a high-level of abstraction for their debugging environment, but have limited performance in terms of speed and no in-situ capacity. Software gate-level simulators (such as QuickSim™ from Mentor Graphics Corporation) typically offer limited level of abstraction and speed as well as no in-situ capacity. Hardware gatelevel simulators (such as Cobalt™ and System Realizer™ from Quickturn Inc., Avatar™ from Ikos, and fast-prototyping systems usually built from FPGAs) typically offer very good performance in terms of speed and in-situ capacity, but a limited debugging environment.

When testing the design described by the HDL source code a designer may choose to simulate and validate the design at the RTL source code level (i.e., RTL simulation). RTL simulation typically permits the designer to set breakpoints in the source code, navigate the design hierarchy, view variables and signals and trace the value of these variables and signals.

When testing complex designs, millions or billions of test vectors may need to be applied in order to adequately test the design. Hardware accelerators or emulators can be used with the gate-level design to test the design at a much greater speed than what is typically possible through software simulation (i.e. either software RTL simulation or software gate-level simulation). Unfortunately, the gate-level design generated in step 150 typically includes none of the high-level information available in the RTL source code **110**. As a result, features available during RTL simulation such as setting breakpoints or analyzing the source code coverage are not available during gate-level simulation.

Instrumentation is the process of preserving high-level information through the synthesis process. Instrumentation permits simulation of a gatelevel netlist at the level of abstraction of RTL simulation by preserving some of the information available at the source code level through the synthesis process.

FIG. **2** illustrates one embodiment of the instrumentation process in which instrumentation is integrated with the synthesis process. RTL source code **210** is provided to the synthesis process **220**. The synthesis process **120** of FIG. **1** has been modified to include an instrumentation step **234**. After instrumentation the instrumented code is then synthesized in step **240** as the original RTL source code was in step **140** of FIG. **1**.

In one embodiment, instrumentation results in generating a modified gate-level design to permit reconstitution of the flow of execution of the original RTL source code during gate-level simulation. Generally instrumentation logic is created for a synthesizable statement in the RTL source code either by modifying the RTL source code or by analyzing the RTL source code during the synthesis process. The instrumentation logic provides an output signal indicative of whether the corresponding synthesizable statement is active. A gate-level design including the instrumentation output signal is then synthesized. Referring to FIG. **2**, the resulting gate-level design **250** contains additional logic to create the additional instrumentation output signals referenced in instrumentation data **238**.

In an alternative embodiment, the RTL source code is analyzed to generate a cross-reference database as instrumentation data **238** without modifying the gate-level design. The cross-reference database indicates the combination of already existing signals in the form of instrumentation logic that can be evaluated during simulation to determine whether a particular line of the RTL source code is active. The cross-reference database contains a cross-reference between these instrumentation logic output signals and the position of the corresponding statement in the source code. The instrumentation data **238** is likely to contain considerably more complex logic to evaluate during simulation when the approach of not modifying the gate-level design (i.e., “pure” cross-reference database) is taken.

The two approaches have tradeoffs. The gate-level design modification technique does not require special knowledge of the target simulation environment. Moreover, the gate-level design modification technique significantly reduces or eliminates the complexity of the logic to be evaluated during simulation to the extent that emulator or accelerator hardware triggering circuitry can be used to take an action when the corresponding statement is executed.

For example, the hardware triggering circuitry may be used to halt the simulation at a particular statement or to count the number of times a particular statement is executed. The resulting gate-level design used during simulation, however, will not be the design actually used for production thus simulation may not verify accurately the behavior of the gate-level design used for production. Furthermore, simulation of modified gate-level design may require more physical resources in hardware than the original design alone if gates have been added in order to implement the instrumentation logic.

Alternatively, the pure cross-reference database technique typically results in greater complexity of instrumentation logic to evaluate during simulation, but does not otherwise affect the original gate-level design. The greater complexity, however, may prevent the use of the hardware triggering circuitry to halt the simulation or to track source code coverage. Thus the pure cross-reference database technique may result in a significantly slower simulation time. Furthermore, since the evaluation may be performed by software, direct verification of the gate-level design in the target system through in-situ verification may not be possible. The instrumentation data including the logic added for instrumentation purposes can be eliminated after testing, however, without disrupting the gate-level design.

In essence the gate-level design modification technique greatly simplifies the analysis and the instrumentation logic required for cross-referencing by modifying the gate-level design to create unique signals and therefore simpler logic to evaluate (i.e., a single signal). The resulting instrumentation logic cross-referenced in the instrumentation data **238** is easily evaluated during simulation. Various embodiments of instrumentation may combine the gate-level design modification technique or the pure cross-referencing technique in order to trade off simulation speed, density, and verification accuracy.

If the gate-level simulator, hardware accelerator, or emulator (e.g., through the use of a logic analyzer which can be external to the emulator) has the capacity to set breakpoints whenever certain signals reach a given value, then it is possible to implement breakpoints corresponding to RTL simulation breakpoints in the gate-level design. Whenever the user specifies a breakpoint in the RTL source code, the condition can be converted to a comparison with key signals in the gate-level design.

Instrumentation data **238** identifies the RTL source code statements each instrumentation output signal is associated with. Instrumentation data **238** is generated during the instrumentation process of step **234**. In one embodiment, the instrumentation data is implemented as gates that can then be simulated by the target-level simulator. By examining the state of each instrumentation output signal during gate-level simulation, the user can determine which portions of RTL source code are being simulated. This in turn permits the designer to determine RTL source code coverage. By tracking the instrumentation signal values for each cycle of execution, the designer can determine how many times each line of the RTL source code has been activated.

The instrumentation data **238** can be used during simulation to ensure every possible state transition has been tested. For example, a Finite State Machine analyzer can determine from the values of the instrumentation output signals whether every possible state transition has been tested.

The instrumentation data **238** can also be used to enhance the source code display. In one embodiment, the source code is repositioned on the display so as to indicate the execution paths that are active during a current cycle. In another embodiment, the active source code in a given cycle is highlighted to indicate that it is active. This permits the designer to visually see the process flow without having to determine the value of each signal. In one embodiment, the instrumentation data **238** is used to enhance the display of the original RTL source code rather than the source code resulting from instrumentation.

An integrated circuit design is typically built by assembling hierarchical blocks. In VHDL, a block corresponds to an entity and architecture. In Verilog, a block corresponds to a module. In both HDLs, a block typically includes a declarative portion and a statement portion. The declarative portion generally includes the list of the ports or connectors.

The statement portion describes the block's behavior and is typically where a designer needs help when debugging a design. The statement portion includes concurrent signal assignment statements and sequential statements.

Concurrent signal assignment statements assign a logic expression to a signal. The signal is typically available for viewing at all times and thus breakpoints can be set in accordance with when the signals reach a certain value.

Sequential statements assign values depending upon the execution flow of the sequence. Sequential statement analysis is typically where the designer needs the greatest aids in debugging the design.

Sequential statements are typically found in VHDL "processes" and in Verilog "always" blocks. Processes or always blocks can be built of an unlimited combination of sequential statements including loops, conditional statements, and alternatives. There are at least two classes of sequential statements: level-sensitive and event-sensitive. Level-sensitive sequential statements only depend on the value of the inputs and can be synthesized to logic networks of combinatorial gates and latches. Event-sensitive sequential statements additionally require sequential logic such as flip-flops.

In one embodiment, level-sensitive RTL source code is instrumented by creating and associating one output signal with each list of synthesizable sequential statements. A list can consist of one or more sequential statements.

In one embodiment, each statement is a list. In an alternative embodiment, each list corresponds to a branch of the RTL source code. A list corresponding to a branch typically comprises a plurality of adjacent sequential statements, but may comprise a single sequential statement. Only one output signal is needed for each list of synthesizable sequential statements in a branch rather than for every sequential statement in the source code. Examples of sequential statements that create branches in the RTL source code are conditional statements such as IF-THEN statements and SELECT-CASE statements.

FIG. **3** illustrates one method of modifying RTL source code for level-sensitive code. Generally, a unique local variable is created for each list of adjacent sequential statements in step **310**. The level sensitive code instrumentation includes the step of modifying the RTL source code to initialize each of these unique variables to zero at the beginning of the process being instrumented in step **320**. One unique variable assignment statement is inserted into each list of adjacent sequential statements corresponding to an executable branch in step **330**. The assignment statement sets the unique variable to one. At the end of the process all the unique local variables are assigned to global signals in step **340**. Steps **310** and **320** are more generically referred to as initialization. Step **330** is referred to as flow instrumentation. Step **340** is referred to as "gathering."

FIG. **4** illustrates non-instrumented VHDL source code. The VHDL source code **400** includes nine sequential statements within the process block. Eight of these nine statements are non-signal assignment sequential statements. These eight sequential statements form six statement lists or executable branches of the code. IF-THEN statement **410** comprises one list. Signal assignment statement **420** comprises a second list. Statements **430**, **440**, **450** and **490** comprise a third list because they would be executed sequentially within the same execution path. Statements **460**, **470**, and **480** form individual lists.

FIG. **5** illustrates one embodiment of the logic **500** resulting from the synthesis of the RTL source code of FIG. **4**. This figure may be used for comparison with the gate level design generated from instrumented code described below.

FIG. **6** illustrates the source code of FIG. **4** after instrumentation as described in FIG. **3**. The added statements are italicized for emphasis. For example, line **612** has been added to the source code to create six unique local variables (TRACE1 through TRACE6), one for each of the six identified lists, in accordance with step **310** of FIG. **3**.

In accordance with step **330** of FIG. **3**, a trace variable assignment statement has been added adjacent to each of the lists. Referring to FIGS. **4** and **6**, variable assignment statement **630** has been added adjacent to the first list comprising statement **410**. Variable assignment statement **632** has been added adjacent to the second list comprising statement **420**. Variable assignment statement **634** has been added adjacent to the third list comprising statements **430**, **440**, **450** and **490**. Variable assignment statement **636** has been added adjacent to the fourth list comprising statement **460**. Similarly, variable assignment statements **638** and **640** have been added adjacent to the fifth list comprising statement **470** and the sixth list comprising **480**, respectively. Each of variable assignment statements **630** through **640** assigns a unique local variable the value of one.

Code portion **620** is added to initialize the unique local variables to zero at the beginning of the process in accordance with step **320** of FIG. **3**.

Each of the local variables is assigned to a global output signal in accordance with step **340** of FIG. **3** by code portion **650**. If required by the HDL, the global signals are declared by code portion **610**. Similarly, the trace variables are declared by code portion **612**.

In one embodiment, the unique local variables can actually be a single array where each "unique variable" or trace variable corresponds to a different position in the array. Similarly, in one embodiment, the additional global signals are described by an array where each of the global signals is represented by a different index of the array.

Coding practices for VHDL generally require variables to be used within the process and a signal assignment at the end of the process to propagate the variable values at the end of the process. In one embodiment, markers such as variable assignment statements are used to track the execution paths. Markers such as variable assignment statements are not typically synthesized into logic indicating the variable values, thus the variable assignment statements are used in conjunction with signal assignment statements in order to produce signals indicating whether various portions of the synthesized code are being executed.

If permitted by the HDL, however, global signal assignments can be used in lieu of local variable assignment statements. This would simplify the process of FIG. **3** in that there would be no need to create or initialize local variables. In addition the step of assigning the local variables to global signals could be eliminated because values are assigned directly. The key is ensuring that there is a unique output signal created and associated with each list of sequential statements regardless of the coding practice used to achieve this goal.

FIG. **7** illustrates one embodiment of the logic **700** generated through instrumentation. In particular, FIG. **7** illustrates the additional gate-level logic added to generate signals SIG_TRACE1 through SIG_TRACE6 from synthesis of the modified source code.

FIG. **8** illustrates a Verilog "always" block **800**. FIG. **9** illustrates the same code after instrumentation in accordance with the process of FIG. **3**. Due to Verilog syntax requirements, "BEGIN-END" statements were used to properly group the instrumentation variable with the other statements in each executable path.

Although the code of FIG. **8** results in a latch, application of the technique of FIG. **3** to the source code of FIG. **8** ensures that the instrumentation output signals are the result of combinatorial logic only. Thus the logic for determining which lines of code are active can be purely combinatorial even when the RTL source code is synthesized into latches.

FIG. **10** illustrates one embodiment of gate-level logic **1100** generated by synthesis of the instrumented "always" block **900** of FIG. **9**. The instrumentation signals SIG__TRACE1, SIG__TRACE2, SIG__TRACE3, and SIG__TRACE4 are the result of combinatorial logic only.

Referring to FIG. **2**, the instrumentation data **238** can be stored in a cross-reference file. In one embodiment, the cross-reference file contains a mapping between original source code line numbers and instrumentation signals. Each time an instrumentation variable (and its associated signal) is added to the source code, all the line numbers of the statements in the list associated with the instrumentation variable are added to the file. This cross-reference file (i.e., instrumentation data **238**) can be used by the gate-level simulation environment to convert the designer's breakpoints into actual conditions on instrumentation signals.

A more sophisticated method than that illustrated in FIG. **3** is required to instrument RTL source code having references to signal events. Typically such source code is used to describe edge-sensitive devices. References to signal events typically imply flip-flops. A signal event is a signal transition. Thus any signal computed from a signal transition references a signal event.

FIG. **11** illustrates sample VHDL code **1100** with references to a signal event. VHDL code **1100** implements a D-type flip-flop with asynchronous reset. The event in this example is a transition on the clock signal (CLK) as referenced by the term "CLK'EVENT."

In accordance with VHDL specifications signals can have various attributes associated with them. A function attribute executes a named function on the associated signal to return a value. For example, when the simulator executes a statement such as CLK'EVENT, a function call is performed to check this property of the signal CLK. In particular, CLK'EVENT returns a Boolean value signifying a change in value on the signal CLK. Other classes of attributes include value attributes and range attributes.

In VHDL code **1100**, the signal CLK has a function attribute named "event" associated with it. The predicate CLK'EVENT is true if an event (i.e., signal transition) has occurred on the CLK signal. Assigning a value to a signal (i.e., a signal transaction) qualifies as an event only if the transaction results in a change in value or state for the signal. Thus the predicate CLK'EVENT is true whenever an event has occurred on the signal CLK in the most recent simulation cycle. The predicate "IF (CLK'EVENT and CLK='1')" is true on the rising edge of the signal CLK.

Depending upon the specifics of the HDL, another function such as RISING__EDGE(CLK) might be used to accomplish the same result without the use of attributes. The function RISING__EDGE(CLK) is still an event even though the term "event" does not appear in the function.

FIG. **12** illustrates a method of instrumenting source code having references to signal events. In step **1210**, every signal event is sampled using a fast clock. In other words, every signal whose state transition serves as the basis for the determination of another signal is sampled. An instrumentation signal event corresponding to the original signal event is generated in step **1220**. Any attributes of the original signal must similarly be reproduced based on the instrumentation signal if the source code uses attributes of the original signal event.

In step **1230**, every process that references a signal event is duplicated. In step **1240**, each list of sequential statements within the duplicate version of the code is replaced by a unique local variable assignment statement. In step **1250**, each time a signal event is referenced in the duplicated version of the code, it is replaced by the sampled signal event computed in step **1210**. The modified RTL source code can then be synthesized in step **1260** to generate gate-level logic including the instrumentation output signals.

FIG. **13** illustrates application of the method of FIG. **12** to the source code of FIG. **11**. In order to detect signal events properly for instrumentation, the signal events are sampled using a fast clock provided during gate-level simulation (i.e., FAST__CLK). FAST__CLK has a higher frequency than the CLK signal and thus permits detecting transition edges before signals depending upon CLK (including CLK itself) can.

The only signal event referenced in FIG. **11** is a transition in the signal CLK indicated by the term CLK'EVENT. Thus an instrumentation version of CLK'EVENT is created by sampling the signal CLK using FAST__CLK. The signal FAST__CLK has a higher frequency than the signal CLK.

Code portion **1310** samples the CLK signal on every rising edge of the signal FAST__CLK to generate a sampled version of the signal CLK named SAMPLED__CLK. The instrumentation version of CLK'EVENT is CLK__EVENT which is generated in code portion **1310** based on SAMPLED__CLK. The instrumentation signal CLX__EVENT (corresponding to CLK'EVENT) is determined by comparison of signals SAMPLED__CLK and CLK. The signal CLK__EVENT is true only when the signal SAMPLED__CLK is not the same as CLK, thus indicating a transition has occurred in the signal CLK.

Although not required for this example, code portion **1310** also illustrates the generation of instrumentation clock signal attributes based on SAMPLED__CLK. For example, the signal CLK'STABLE is the complement of CLK'EVENT. Thus code portion **1310** indicates the instrumentation version of the attribute CLK'STABLE (i.e., CLK__STABLE) computed on the instrumentation clock signal (i.e., SAMPLED__CLK). The signal CLK'LASTVALUE is a function signal attribute that returns the previous value of the signal CLK. The instrumentation version (i.e., CLK__LASTVALUE) of the attribute CLK'LASTVALUE is similarly computed on the instrumentation clock signal SAMPLED__CLK.

Although CLK__LASTVALUE is the same as the sampled clock signal, SAMPLED__CLK, code **1310** introduces the intermediate signal SAMPLED__CLK for purposes of illustrating sampling of the CLK signal. The signal CLK__LASTVALUE can be defined in lieu of SAMPLED__CLK in order to eliminate the introduction of an unnecessary intermediate signal SAMPLED__CLK and the subsequent step of assigning CLK__LASTVALUE the value of SAMPLED__CLK.

Neither CLK__LASTVALUE nor CLK__STABLE are needed in this example for code portion **1320**, however, code portion **1310** serves as an example of how to generate instrumentation versions of signal attributes typically used to describe edge-sensitive devices.

Code portion **1320** represents the instrumented duplicate of original code portion **1330**. The process of code portion

1330 references the event CLK'EVENT in the IF-ELSIF statement. In code portion **1320**, all sequential statements (except the statement referencing an event) have been replaced with unique local variable assignment statements. These statements assign a local variable (i.e., TRACE1, TRACE2) the value "1." Code portion **1320** also includes statements to create and initialize these unique local variables.

In accordance with step **1240**, every occurrence of a signal event is replaced with the sampled version of that event. Thus, for example, references to CLK'EVENT in code portion **1330** are replaced with references to CLK__EVENT in code portion **1320**. Moreover, the process parameter list is modified to include the generated signal CLK__EVENT. FIG. **14** illustrates the gate-level logic **1400** resulting from synthesis of the code in FIG. **13**.

FIG. **15** illustrates Verilog source code **1500** for a D flip-flop with asynchronous reset. FIG. **16** illustrates the code **1600** resulting from modifying source code **1500** in accordance with the method of FIG. **12**.

One advantage of the instrumentation approach of FIG. **12** is that the gates generated by the synthesis tool are the same ones that would be generated if the source code had not been instrumented. The gates generated for the instrumentation logic are not intermingled with the gates generated from the non-instrumented source code. This permits design verification with gate-level logic that does not need to be re-verified after instrumentation verification. Thus the designer can verify the result of synthesis at the gate level while retaining RTL breakpoint feature. In some cases, however, the synthesis tool may not recognize that the same code appears twice. This may incur an additional relatively expensive phase of resource sharing in order to achieve the same performance results as the process illustrated in FIG. **3**.

One advantage of the instrumentation process of FIG. **3** over that of FIG. **12**, however, is that a synthesis tool can typically analyze the source code to detect obvious resource sharing.

The instrumentation methods of FIGS. **3** and **12** permit detecting any path that has been taken while a VHDL process or a Verilog "always" block is active. Tracking the activation of each process permits further analysis.

FIG. **17** illustrates a method of instrumenting the activation of the processes (or "always" blocks) themselves for subsequent determination of whether the process is active during gate-level simulation .

In step **1710**, the sensitivity list of a process is identified. In step **1720**, logic is generated to compare the signals in the sensitivity list between consecutive simulation cycles. Subsequently, during gate-level simulation in step **1730**, a determination is made as to whether an event has occurred on any of the sensitivity list signals. Each simulation cycle that a signal indicates a difference (i.e., a signal event has occurred), the process is active as indicated by step **1740**. Otherwise, if no events have occurred on any of the sensitivity list signals, the process is inactive as indicated by step **1750**.

FIG. **18** illustrates the code added to determine if process **P1** is active. The added code is italicized. The sensitivity list of process **P1** includes signals a, b, and c. In accordance with step **1720** of FIG. **17**, code section **1810** creates sampled versions of a, b, and c using FAST__CLK as described above. The sampled versions of a, b, and c are SAMPLED__A, SAMPLED__B, and SAMPLED__C, respectively.

Code section **1820** determines if an event has occurred on each of the sensitivity list signals. The test "(SAMPLED__A /=A)" is true if an event occurs with respect to signal A. Similarly "(SAMPLED__B /=B)" and "(SAMPLED__C /=C)" indicate whether an event has occurred with respect to signals B and C. Process **P1** is active if any one of these tests is true. Thus the variable P1__ACTIVE is generated by combining each of these signal events using the logical OR function in code section **1820**. Thus signal P1__ACTIVE indicates whether process **P1** is active.

Process instrumentation data can be added to the instrumentation data cross-reference file in order to enhance the source code display. For example, the active process in a given cycle can be highlighted to indicate it is active. This permits the designer to visually see the active processes without having to determine the value of each signal. In one embodiment, the instrumentation data is used to enhance the display of the original RTL source code rather than the source code resulting from instrumentation.

The instrumentation techniques presented result in gate level designs providing explicit instrumentation signals to indicate that some specific portion of the source code is active. The number of instrumentation signals tends to increase with the complexity of the system being modeled.

Some optimizations may be performed to decrease the number of instrumentation signals. At least one execution path will be active any time a process is activated. As a result, the TRACE1 variable in the examples of FIGS. **6** and **9** tend to provide no additional information and thus SIG__TRACE1 is somewhat trivial as can be seen from the synthesized logic of FIGS. **7** and **10**. Thus at least one trace variable (and therefore one output signal trace) can typically be eliminated.

In some cases the execution status of each branch of the code can be determined even though every branch is not explicitly instrumented. To verify the execution status of every branch, the instrumentation process need only ensure that each branch is instrumented either explicitly or implicitly through the instrumentation of other branches.

In some instances, the capacity of hardware triggers can be used to eliminate some of the instrumentation by combining several signals into one condition. The number of gates simulated can be reduced by replacing logical AND conditions that appear in the equations of instrumentation signals by simulator-specific triggers.

For example, consider the instrumented CASE statement code fragment **1910** illustrated in FIG. **19**. For purposes of example, only the trace variable assignment statements are shown for the four possible cases. A synthesis tool will generate four comparisons with the vector "opcode." Each trace variable is associated with one of the possible values of opcode. Clearly, however, the additional logic is unnecessary because setting a breakpoint on any one of the case conditions corresponds to setting a trigger on the vector for the corresponding value of "opcode."

FIG. **20** illustrates a method for optimizing the instrumentation process. In particular, an instrumentation signal is selected in step **2010**. In step **2020**, a determination is made to whether the equation of the current signal can be expressed as a logical AND between a signal and a simplified expression. If so, then the AND gate should be eliminated in step **2030** and the extracted signal can be added to the trigger conditions during simulation in step **2040**. If triggers can be activated on zeroes as well as ones, then step **2020** can also determine whether an equation can be simplified as a logical negation of a subexpression and the logical negation of the subexpression can be added to the trigger conditions during simulation in step **2040** where appropriate. Step **2020** would then be applied recursively until the equation cannot be further simplified. This process is then applied to all of the instrumentation signals.

For example, signal TRACE**4** is the result of performing a logical AND between opcode(**0**) and opcode(**1**). Thus TRACE**4** is active only when opcode ="11". In accordance with FIG. **20**, the AND gate can be removed and the simulator trigger conditions would be changed from TRACE**4**=1 to "OPCODE(**0**)=1 AND OPCODE(**1**)=1." This process would then be applied recursively to all signals remaining in the trigger condition. Thus if OPCODE(**0**) happened to be the result of an AND between two other signals, the AND gate could again be eliminated from the synthesized gate-level design and the trigger conditions could be updated accordingly as long as no other signals used "OPCODE(**0**)" as an input. If no other logic uses "OPCODE(**0**)" as an input, then the trigger conditions can be updated to refer to the signals used to generate OPCODE (**0**) and the gate-level netlist AND gate can safely be eliminated. More generally, any optimization that consists of eliminating gates and other elements by transferring the implementation of the instrumentation logic to the logic analyzer of the target simulator can be performed.

Where permitted by the gate-level simulator, the instrumentation required for detecting activation of a process may similarly be reduced. In particular, greater efficiency may be possible by keeping a list of all the signals in the process sensitivity list and then testing whether events occurred on the signals in the sensitivity list. Further optimization may be made possible by sharing the logic for signals that appear on the sensitivity list of more than one process. The original signal can be sampled once initially. A comparison is made between the initial value and the current value of the signal to generate an event signal indicative of whether an event has occurred on that signal. The event signal can then be used for instrumentation of processes with events and for tracking process activation.

FIGS. **3**, **12**, and **17** illustrate methods of modifying the original RTL source code for instrumenting processes and level-sensitive and edge-sensitive source code. Trace variables (i.e., instrumentation variables) can be used to track the execution of any path within the source code. Additional output signals are generated from instrumentation variables in order to detect the execution paths of the source code. In the illustrated embodiments, the instrumentation variables are reset at the beginning of a process and the signals are assigned at the end of the process in order to ensure that all the signals are assigned regardless of which execution path is taken inside the process.

In an alternative embodiment, the signals might be directly assigned in the execution path of the process. Typically, this alternative embodiment would force the synthesis tool to generate complicated structures including latches due to the nature of HDLs and simulation rules.

The methods of FIGS. **3**,**12**, and **17** can be applied to the source code before the source code is synthesized. Thus in one embodiment the steps that modify the RTL source code can be performed before but entirely independently of the synthesis process itself.

FIG. **21** illustrates an embodiment in which the instrumentation data is generated entirely within the synthesis process. The process of creating output signals associated with synthesizable statements in the source code and then synthesizing the source code into a gate-level design including the output signal can be incorporated into the synthesis tool itself so that modification of the RTL source code is not required.

For example, one of the steps performed by a synthesis tool for generation of the gate-level design is parsing the RTL source code. Parsing the RTL source code results in a parser data structure that is subsequently used to generate the gate-level design. Instead of modifying the source code, the synthesis tool can simply set markers inside the parser data structure.

FIG. **22** illustrates one application of using the instrumentation signals for tracing execution flow using breakpoints. In step **2210**, the user sets a breakpoint at a specified line number of the source code. The specified line number is then associated with one of the instrumented lists of statements in step **2220**. In step **2230**, the instrumentation signal for the associated list is identified as the breakpoint output signal.

During the gate-level simulation run, the active lists (identified by transitions in their corresponding instrumentation signals) may be highlighted and displayed for the user as indicated in step **2240**. For example, the active lists may be portrayed in a different color than the inactive lists. Alternatively, the active lists may be displayed using blinking characters, for example. The instrumentation data file can be used to associate an instrumentation signal with a list of source code line numbers to be highlighted.

In response to a 0 to 1 transition in the breakpoint output signal, the simulation can be stopped as indicated in step **2250**. Thus through instrumentation the designer has the ability to effectively set breakpoints in the RTL source code which can be acted upon during RTL simulation.

The methods of instrumentation may be implemented by a processor responding to a series of instructions. In various embodiments, these instructions may be stored in a computer system's memory such as random access memory or read only memory.

The instructions may be distributed on a nonvolatile storage medium for subsequent access and execution by the processor. Typically the instructions are stored in the storage medium for distribution to a user. The instructions may exist in an application program form or as a file stored in the storage medium. The instructions are transferred from the nonvolatile storage medium to a computer system for execution by the processor.

In one embodiment, the program or file is installed from the storage medium to the computer system such that the copy of the instructions in the nonvolatile storage medium is not necessary for performing instrumentation. In another embodiment, the program or file is configured such that the original nonvolatile storage medium is required whenever the instructions are executed.

Nonvolatile storage mediums based on magnetic, optical, or semiconductor memory storage principles are readily available. Nonvolatile magnetic storage mediums include floppy disks and magnetic tape, for example. Nonvolatile optical storage mediums include compact discs, digital video disks, etc. Semiconductor-based nonvolatile memories include rewritable flash memory.

Instrumentation allows the designer to perform gate-level simulation of synthesized RTL designs with source-level debugging. In addition, the instrumentation process allows the designer to examine source code coverage during simulation.

In the preceding detailed description, the invention is described with reference to specific exemplary embodiments thereof. Various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method comprising the steps of:

a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.

2. The method of claim **1** wherein step b) includes the step of:

i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.

3. The method of claim **1** wherein step b) includes the steps of:

i) initializing a marker to a first value at the beginning of a process within the source code; and

ii) setting the marker to a second value within a same executable branch of the source code as the identified statement.

4. The method of claim **3** further comprising the step of:

iii) assigning the value of the marker to the instrumentation signal at the end of the process.

5. A method of generating a gate level design, comprising the steps of:

a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level design having the instrumentation signal.

6. The method of claim **5** wherein step a) further comprises the step of:

i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and

ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

7. The method of claim **6** wherein the variable is assigned a first value in step a)i), the method further comprising the step of:

iii) modifying the source code to initialize the unique variable to a second value.

8. The method of claim **5** wherein step a) is repeated to create a unique instrumented output signal for each list of sequential statements in the source code, wherein each list corresponds to a synthesizable executable branch of the source code.

9. The method of claim **5** further comprising the step of:

c) generating cross-reference instrumentation data mapping each statement in a selected list to the instrumented output signal associated with that list for every list in the source code.

10. The method of claim **9** further comprising the steps of:

d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.

11. The method of claim **5** further comprising the steps of:

c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.

12. A method of generating a gate-level netlist, comprising the steps of:

a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;

b) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;

c) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and

d) synthesizing the source code into a gate-level design including the instrumentation signals.

13. The method of claim **12** wherein step b) further comprises the steps of:

i) assigning each unique local variable a first value; and

ii) initializing each local variable with second value.

14. The method of claim **12** further comprising the step of

e) mapping every statement within a selected list to the corresponding instrumentation signal for that selected list as cross-reference instrumentation data.

15. The method of claim **12** further comprising the steps of:

e) setting a breakpoint at a selected statement of the source code;

f) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and

g) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in the breakpoint signal transitioning to a pre-determined value.

16. A method of generating a gate level netlist, comprising the steps of:

a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;

b) modifying the source code to generate a corresponding sampled version of each signal event in a selected process;

c) modifying the source code to duplicate the selected process;

d) replacing each occurrence of a selected signal event with the corresponding sampled version in the duplicated process;

e) replacing each list of sequential statements within an executable branch of the duplicated process with a unique variable assignment statement;

f) modifying the duplicated process to include an instrumentation signal assignment for each unique variable; and

g) synthesizing the modified source code into a gate-level design.

17. The method of claim **16** wherein step e) further comprises the steps of:

i) assigning the unique variables a first value; and

ii) initializing the unique variables with second value.

18. The method of claim **16** further comprising the step of

e) mapping every statement within each selected list to its corresponding instrumentation signal.

19. The method of claim **16** further comprising the steps of:

h) setting a breakpoint at a selected statement of the source code;

i) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and

j) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the breakpoint signal to a predetermined value.

20. A method of debugging a gate-level design including the steps of:

a) setting a breakpoint at a selected statement of a register transfer level (RTL) synthesizable source code;

b) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements, wherein the list corresponds to an executable branch of the source code including the selected statement;

c) modifying the source code to include an instrumentation signal assignment statement for the local variable; and

d) generating a gate-level design from the modified source code.

21. The method of claim **20** further comprising the steps of:

e) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the instrumentation signal to a pre-determined value.

22. The method of claim **20** wherein step b) further comprises the steps of:

i) assigning the local variable a first value; and

ii) initializing the local variable with second value.

23. The method of claim **20** further comprising the step of

e) mapping every statement within the executable branch of source code to the instrumentation signal.

24. A method of simulating a gate-level design comprising the steps of:

a) identifying a sensitivity list of a process;

b) generating logic to identify signal events for any signal in the sensitivity list; and

c) identifying the process as active during simulation when a signal event occurs for any signal in the sensitivity list.

25. The method of claim **24** wherein step c) further comprises the step of:

i) highlighting a source code description of the process displayed during simulation.

26. The method of claim **24** wherein step b) further comprises the step of:

i) sampling each signal in the sensitivity list to generate corresponding instrumented signals; and

ii) comparing each signal in the sensitivity list with its corresponding instrumented signal to test each signal in the sensitivity list for an event.

27. The method of claim **26** wherein step c) further comprises the step of:

i) generating an active process output signal defined by logically ORing the results of the comparisons.

28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.

29. The storage medium of claim **28** wherein the processor performs the steps of:

i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and

ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

30. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to perform the steps of:

a) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;

b) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and

c) synthesizing the source code into a gate-level design including the instrumentation signals.

31. The storage medium of claim **30** having stored therein further instructions to enable the processor to perform the step of:

d) mapping every statement within each selected list to its corresponding instrumentation signal.

32. A storage medium having stored therein processor executable instructions for debugging a gate level design during simulation, wherein when a breakpoint is set at a selected statement of a register transfer level (RTL) synthesizable source code the instructions enable the processor to perform the steps of:

a) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements within the source code, wherein the list corresponds to an executable branch of the source code including the selected statement;

b) modifying the source code to include an instrumentation output signal assignment statement for the local variable; and

c) generating a gate-level design from the modified source code.

33. The storage medium of claim **32** having stored therein further instructions to enable the processor to perform the step of:

d) mapping every statement within each selected list to its corresponding instrumentation signal.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,240,376 B1 Page 1 of 1
DATED : May 29, 2001
INVENTOR(S) : Alain Raynaud and Luc M. Burgun

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 1,
Line 11, "hen" should read -- when --.
Line 17, "modem" should read -- modern --.
Line 37, "simulator. alternatively," should read -- simulator. Alternatively, --.
Line 38, "converting the ate-" should read -- converting the gate- --.
Line 47, "RTh" should read -- RTL --.

Column 2,
Line 39, "RTh" should read -- RTL --.

Column 10,
Line 33, "signal CLX_" should read -- signal CLK_ --.

Signed and Sealed this

Sixteenth Day of July, 2002

Attest:

[illegible]

JAMES E. ROGAN

Attesting Officer

Director of the United States Patent and Trademark Office

U.S. Patent No. 7,069,526

US007069526B2

(12) **United States Patent**
Schubert et al.

(10) **Patent No.: US 7,069,526 B2**
(45) **Date of Patent: *Jun. 27, 2006**

(54) HARDWARE DEBUGGING IN A HARDWARE DESCRIPTION LANGUAGE

(75) Inventors: **Nils Endric Schubert**, Sunnyvale, CA (US); **John Mark Beardslee**, Menlo Park, CA (US); **Douglas L. Perry**, San Ramon, CA (US)

(73) Assignee: **Synplicity, Inc.**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/027,052**

(22) Filed: **Dec. 29, 2004**

(65) **Prior Publication Data**

US 2005/0125754 A1 Jun. 9, 2005

Related U.S. Application Data

(63) Continuation of application No. 10/406,732, filed on Apr. 2, 2003, now Pat. No. 6,904,577, which is a continuation of application No. 09/724,702, filed on Nov. 28, 2000, now Pat. No. 6,581,191.

(60) Provisional application No. 60/230,068, filed on Aug. 31, 2000, provisional application No. 60/168,266, filed on Nov. 30, 1999.

(51) **Int. Cl.**
G06F 17/50 (2006.01)

(52) **U.S. Cl.** **716/4**; 716/5

(58) **Field of Classification Search** 716/4–8; 703/14–16; 714/724

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,306,286	A	12/1981	Cocke et al.
4,590,581	A	5/1986	Widdoes, Jr.
4,635,218	A	1/1987	Widdoes, Jr.
4,675,646	A	6/1987	Lauer

(Continued)

FOREIGN PATENT DOCUMENTS

DE	4 042 262	7/1992

OTHER PUBLICATIONS

Kirkovski, D., et al., "Improving the Observability and Controllability of Datapaths for Emulation-Based Debugging", IEEE Trans. on CAD, vol. 18, Nov. 1999, pp. 1529-1541.

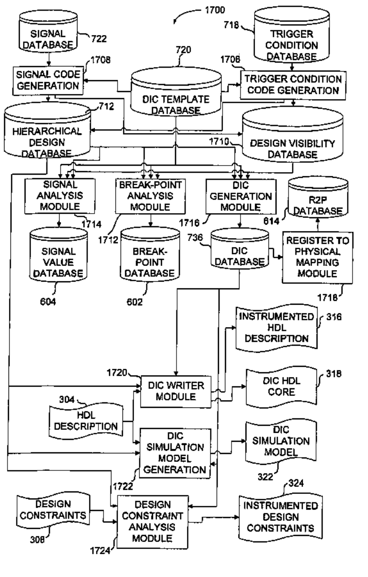
(Continued)

Primary Examiner—Stacy A. Whitmore
Assistant Examiner—Binh Tat
(74) *Attorney, Agent, or Firm*—Blakley, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

Techniques and systems for analysis, diagnosis and debugging fabricated hardware designs at a Hardware Description Language (HDL) level are described. Although the hardware designs (which were designed in HDL) have been fabricated in integrated circuit products with limited input/output pins, the techniques and systems enable the hardware designs within the integrated circuit products to be comprehensively analyzed, diagnosed, and debugged at the HDL level at speed. The ability to debug hardware designs at the HDL level facilitates correction or adjustment of the HDL description of the hardware designs.

33 Claims, 26 Drawing Sheets



U.S. PATENT DOCUMENTS

4,845,712	A	7/1989	Sanner et al.	
4,901,259	A	2/1990	Watkins	
4,937,770	A	6/1990	Samuels et al.	
4,937,827	A	6/1990	Beck et al.	
5,036,473	A	7/1991	Butts et al.	
5,146,460	A	9/1992	Ackerman et al.	
5,281,864	A	1/1994	Hahn et al.	
5,321,828	A	6/1994	Phillips et al.	
5,329,470	A	7/1994	Sample et al.	
5,329,471	A	7/1994	Swoboda et al.	
5,369,593	A	11/1994	Papamarcos et al.	
5,412,260	A	5/1995	Tsui et al.	
5,416,919	A	5/1995	Ogino et al.	
5,425,036	A	6/1995	Liu et al.	
5,537,580	A	7/1996	Giomi et al.	
5,544,311	A	8/1996	Harenberg et al.	
5,546,562	A	8/1996	Patel	
5,560,009	A	9/1996	Lenkov et al.	
5,568,437	A	10/1996	Jamal	
5,572,712	A	11/1996	Jamal	
5,574,388	A	11/1996	Barbier et al.	
5,581,742	A	12/1996	Lin et al.	
5,596,587	A	1/1997	Douglas et al.	
5,640,542	A	6/1997	Whitsel et al.	
5,644,515	A	7/1997	Sample et al.	
5,661,662	A	8/1997	Butts et al.	
5,663,900	A	9/1997	Bhandari et al.	
5,717,699	A	2/1998	Haag et al.	
5,748,875	A	5/1998	Tzori	
5,751,735	A	5/1998	Tobin et al.	
5,754,827	A	5/1998	Barbier et al.	
5,757,819	A	5/1998	Segars	
5,771,240	A	6/1998	Tobin et al.	
5,777,489	A	7/1998	Barbier et al.	
5,790,832	A	8/1998	Barbier et al.	
5,801,956	A	9/1998	Kawamura et al.	364/489
5,805,859	A	9/1998	Giramma et al.	
5,812,414	A	9/1998	Butts et al.	
5,812,562	A	9/1998	Baeg	
5,822,564	A	10/1998	Chilton et al.	
5,831,868	A	11/1998	Beausang et al.	364/489
5,870,308	A *	2/1999	Dangelo et al.	716/18
5,870,410	A	2/1999	Norman et al.	
5,905,883	A	5/1999	Kasuya	395/500
5,907,697	A	5/1999	Barbier et al.	
5,933,356	A *	8/1999	Rostoker et al.	703/15
5,937,190	A	8/1999	Gregory et al.	
5,943,490	A	8/1999	Sample	
5,951,696	A	9/1999	Naaesh et al.	
5,960,191	A	9/1999	Sample et al.	
5,963,735	A	10/1999	Sample et al.	
5,991,523	A	11/1999	Williams et al.	395/500.19
5,999,725	A	12/1999	Barbier et al.	
6,009,256	A	12/1999	Tseng et al.	
6,066,022	A	12/1999	Rhim et al.	
6,014,334	A	1/2000	Patel et al.	
6,021,447	A	2/2000	Szeto et al.	
6,026,230	A	2/2000	Lin et al.	
6,041,176	A	3/2000	Shiell	
6,057,706	A	5/2000	Barbier et al.	
6,107,821	A	8/2000	Kelem et al.	
6,132,109	A	10/2000	Gregory et al.	
6,157,210	A	12/2000	Zaveri et al.	
6,182,247	B1	1/2001	Herrmann et al.	
6,182,268	B1	1/2001	McElvain	
6,188,975	B1	2/2001	Gay	
6,191,683	B1	2/2001	Nygaard, Jr.	
6,199,031	B1	3/2001	Challier et al.	
6,202,044	B1	3/2001	Tzori	
6,202,172	B1	3/2001	Ponte	
6,212,490	B1 *	4/2001	Li et al.	703/14
6,212,650	B1	4/2001	Guccione	
6,223,148	B1	4/2001	Stewart et al.	
6,240,376	B1	5/2001	Raynaud et al.	
6,247,147	B1	6/2001	Beenstra et al.	
6,255,836	B1	7/2001	Schwarz et al.	
6,255,845	B1	7/2001	Wong et al.	
6,286,114	B1	9/2001	Veenstra et al.	
6,292,765	B1	9/2001	Ho et al.	
6,301,688	B1	10/2001	Roy	
6,311,317	B1	10/2001	Khoche et al.	
6,334,207	B1 *	12/2001	Joly et al.	716/17
6,336,087	B1	1/2002	Burgun et al.	
6,363,520	B1	3/2002	Boubezari et al.	
6,374,370	B1	4/2002	Bockhaus et al.	
6,377,911	B1	4/2002	Sample et al.	
6,377,912	B1	4/2002	Sample et al.	
6,378,903	B1	4/2002	Whetsel	
6,389,558	B1	5/2002	Herrmann et al.	
6,389,586	B1	5/2002	McElvain	
6,421,813	B1	7/2002	Jeddeloh	
6,425,101	B1	7/2002	Garreau	
6,427,224	B1 *	7/2002	Devins et al.	716/4
6,434,735	B1	8/2002	Watkins	
6,438,735	B1	8/2002	McElvain et al.	
6,449,762	B1	9/2002	McElvain	
6,456,961	B1 *	9/2002	Patil et al.	703/14
6,460,148	B1	10/2002	Veenstra et al.	
6,460,174	B1	10/2002	Carey	
6,463,392	B1	10/2002	Nygaard et al.	
6,470,478	B1	10/2002	Bargh et al.	
6,477,683	B1 *	11/2002	Killian et al.	716/1
6,493,852	B1	12/2002	Narain et al.	
6,499,123	B1	12/2002	McFarland et al.	
6,510,534	B1	1/2003	Nadeau-Dostie et al.	
6,513,143	B1	1/2003	Bloom et al.	
6,564,347	B1	5/2003	Mates	
6,567,932	B1	5/2003	Edwards et al.	
6,571,375	B1 *	5/2003	Narain et al.	716/5
6,581,191	B1 *	6/2003	Schubert et al.	716/4
6,618,584	B1	9/2003	Mann	
6,618,839	B1 *	9/2003	Beardslee et al.	716/4
6,633,838	B1	10/2003	Arimilli et al.	
6,690,398	B1	2/2004	Beck et al.	
6,697,957	B1	2/2004	Wang et al.	
6,704,889	B1	3/2004	Veenstra et al.	
6,571,751	B1	6/2004	Murray et al.	
6,754,760	B1	6/2004	Yee et al.	
6,754,862	B1	6/2004	Hoyer et al.	
2003/0131325	A1 *	7/2003	Schubert et al.	716/4
2004/0025122	A1 *	2/2004	Schubert et al.	716/4

OTHER PUBLICATIONS

Haufe, M., et al., "Ein Debugger fuer ASIC-Prototypen", pp. 10, DASS Dresden Germany, 2000.

Bulent Dervisoglu, "Design for Testability: It is time to deliver it for Time-to-Market," Proceedings of the Internatonal Test Conference, 1999.

Keshava Iyengar Satish, "Tutorial on Design For Testability (DFT): An ASIC Design Philosophy for testability for Chips to Systems," Sixth Annual IEEE International ASIC Conference and Exhibit, 1993.

Jan Liband, "Techniques for Real-Time Debugging," Embedded Systems Programming, vol. 8, No. 4, Apr. 1995.

Dr. Vinod Agarwal, "Embedded IC Test: A New Plateau for DFT," Evaluation Engineering, vol. 38, No. 9, Sep. 1999.

Stephen O'Reilly, "Debugging Drivers with Emulators and Logic Analyzers," Embedded Systems Programming, vol. 11, No. 2, Feb. 1998.

Jack G. Ganssle, "Debuggers for Modern Embedded Systems," Embedded Systems Programming, Nov. 1998.

Brent Miller, "Scan Conversion of ASICs," Circuit Design, vol. 7, No. 2, Feb. 1990.
Jerry Bauer et al., "A Reconfigurable Logic Machine for Fast Event-Driven Simulation," Design Automation Conference Proceedings (DAC), Jun. 1998, pp. 668-671.
Synopsys, Inc., "BSD Compiler" datasheet, www.synopsys.com/products/test/bsd_ds.html.
Synopsys, Inc., "DFT Compiler" Next Generation 1-Pass Test Synthesis Technology Backgrounder, Apr. 2000.
Cynthia Cousineau et al., "Design of a JTAG Based Run Time Reconfigurable System," 7th IEEE Symposium on Field Programmable Custom Computing Machines, 1999.
John Andrews, "An Embedded JTAG, System Test Architecture," Proceedings of ELECTRO, 1994.
Shen XuBang et al., "Design and Implementation of a JTAG Boundary-Scan Interface Controller," Proceedings of the 2nd IEEE Asian Test Symposium, 1993.
"JTAG for system emulation," Electronic Engineering, vol. 65, No. 794, Feb. 1993.
R.P. van Riessen et al., "Design and Implementation of a Hierarchical Testable Architecture using the Boundary Scan Standard," 1st European Test Conference, IEEE, 1989.
Ing-Jer Huang et al., "ICEBERG: An Embedded In-circuit Emulator Synthesizer for Microcontrollers," Proceedings of the 36th Design Automation Conference (DAC), 1999.
Ross Bannatyne, "Debugging Aids for Systems-on-a-Chip," Proceedings of North Con, 1998.
Robert J. Hasslen et al., "A Validation Strategy for Embedded Core ASICS," Proceedings of the Third Annual IEEE ASIC Seminar and Exhibit, Sep. 1990.
Max Baron et al., "A Real-Time Performance Analyzer," VLSI Systems Design, May 4, 1987.
Indradeep Ghosh, "A BIST Scheme for RTL Circuits Based on Symbolic Testability Analysis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, No. 1, Jan. 2000.
Indradeep Ghosh, "A Design-for-Testability Technique for Register-Transfer Level Circuits Using Control/Data Flow Extraction," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, No. 8, Aug. 1998.
Barbara Tuck, "Test-bench tools ease tedious, time-consuming manual efforts," Computer Design, May 1996.
Krzysztof Badzmirowski et al., "Diagnosis of Digital/Analogue Measurement System with Application of Test Bus and Distributed Diagnostic Subsystem," Proceedings of the IEEE Instrumentation and Measurement Technology Conference, 1998.
Stephen Pateras et al., "BIST: A Test & Diagnosis Methodology for Complex, High Reliability Electronics Systems," 1997 IEEE Autotestcon Proceedings.
John Sweeney, "A Method For Using JTAG Boundary Scan For Diagnosing Module Level Functional Failures," Proceedings of WesCon Conference, 1988.
Wen-Jong Fang et al., "A Multi-Level FPGA Synthesis Method Supporting HDL Debugging for Emulation-Based Designs," Proceedings of the Asian and South Pacific Design Automation Conference, 1999.
David Ruimy Gonzales, "Tool Reusable for DSP System Emulation and Board Production Testing," IEEE Technical Applications Conference, Northcon, 1996.
Ray Weiss, "ICEs try to target higher clock rates, more processors," Computer Design, vol. 35, No. 2, Feb. 1996.
Mike Winters, "Using IEEE-1149.1 For In-Circuit Emulation," Proceedings of WesCon, 1994.
Christopher Perez, "Tools for Embedded-Systems Debugging," Dr. Dobbs Journal, Mar. 1993.
Kristen Ahrens, "Test Standard Speeds On-Board Programming," Electronic Design, Nov. 7, 1994.
Oliver Bringmann et al., "Target Architecture Oriented High-Level Synthesis for Multi-FPGA Based Emulation," Proceedings of the Design Automation and Test Conference, DATE, 2000.
Karlheinz Weiβ et al., "Exploiting FPGA-Features during the Emulation of a Fast Reactive Embedded System," Proceedings of the 1999 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ACM, 1999.
Gernot Koch et al., "Breakpoints and Breakpoint Detection in Source Level Emulation," 9th International Symposium on System Synthesis, 1996.
Gernot Koch et al., "Co-Emulation and Debugging of HW/SW-Systems," 10th International Symposium on System Synthesis (ISSS), 1997.
Gernot Koch et al., "System prototyping in the COBRA Project," International Journal Microprocessors and Microsystems, Elsevier Science, vol. 20, No. 3, 1996.
G. Haug et al., "Behavioral Emulation of Synthesized RT-level Descriptions Using VLIW Architectures," 9th International Workshop on Rapid System Prototyping, 1998.
Gernot Koch et al., "System Validation by Source Level Emulation of Behavioral VHDL Specifications," 6th International Workshop on Rapid System Prototyping, 1995.
Gernot Koch, "Interaktives Debugging algorithmischer Hardware-Verhaltensbeschreibungen mit Emulation," Dissertation, 1998.
Peter Clarke, "DATE99: Temento to launch scan insertion tool," EETimes.com, Mar. 4, 1999.
Alexander Miczo, "Digital Logic Testing and Simulation," Harper & Row, Publishers, 1996.
Xilinx, "ChipScope Software and ILA Cores User Manual," v1.1, Jun. 30, 2000.
Gernot Koch et al., "Breakpoints and Breakpoint Detection in Source Level Emulation," ACM Transactions on Design Automation of Electronic Systems, vol. 3, No. 2, 1998.
Samiha Mourad et al., "123 Logic Analyzers", The Engineering Handbook, pp. 1325-1330, CRC Press, Inc., 1996.
PCT International Search Report, re PCT/US 00/32543, Jun. 28, 2001.
U.S. Appl. No. 09/724,840, filed Nov. 28, 2000.
U.S. Appl. No. 09/724,839, filed Nov. 28, 2000.
U.S. Appl. No. 09/724,585, filed Nov. 28, 2000.
Gert Jan Van Rootselaar and Bart Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. *Proceedings of the International Test Conference* (*ITC*), (1999), 11 pages.
Bart Vermeulen and Gert Jan Van Rootselaar. Silicon Debug of a Co-Processor Arrays for Video Applications. *Proceedings of the IEEE International High-Level Design Validation and Test Workshop* (HLDVT), (2000), pp. 47-52.
Gert Jan Van Rootselaar and Bart Vermeulen. Silicon Debug Methodology: System Level Design and Debug of High Performance Embedded Media Systems, part 3, *ICCAD 1999 Embedded Tutorial,* (1999), 31 pages.

* cited by examiner

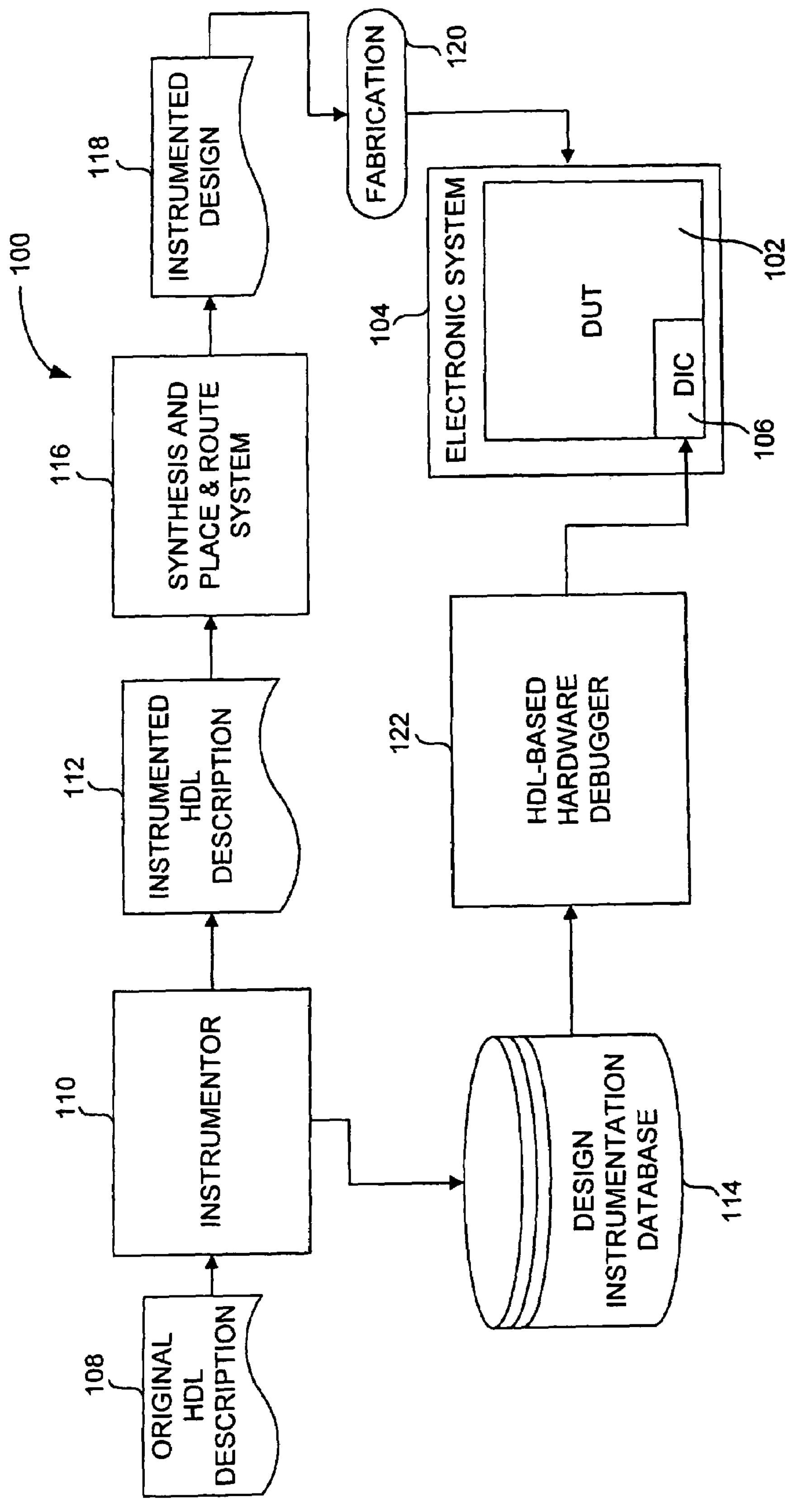
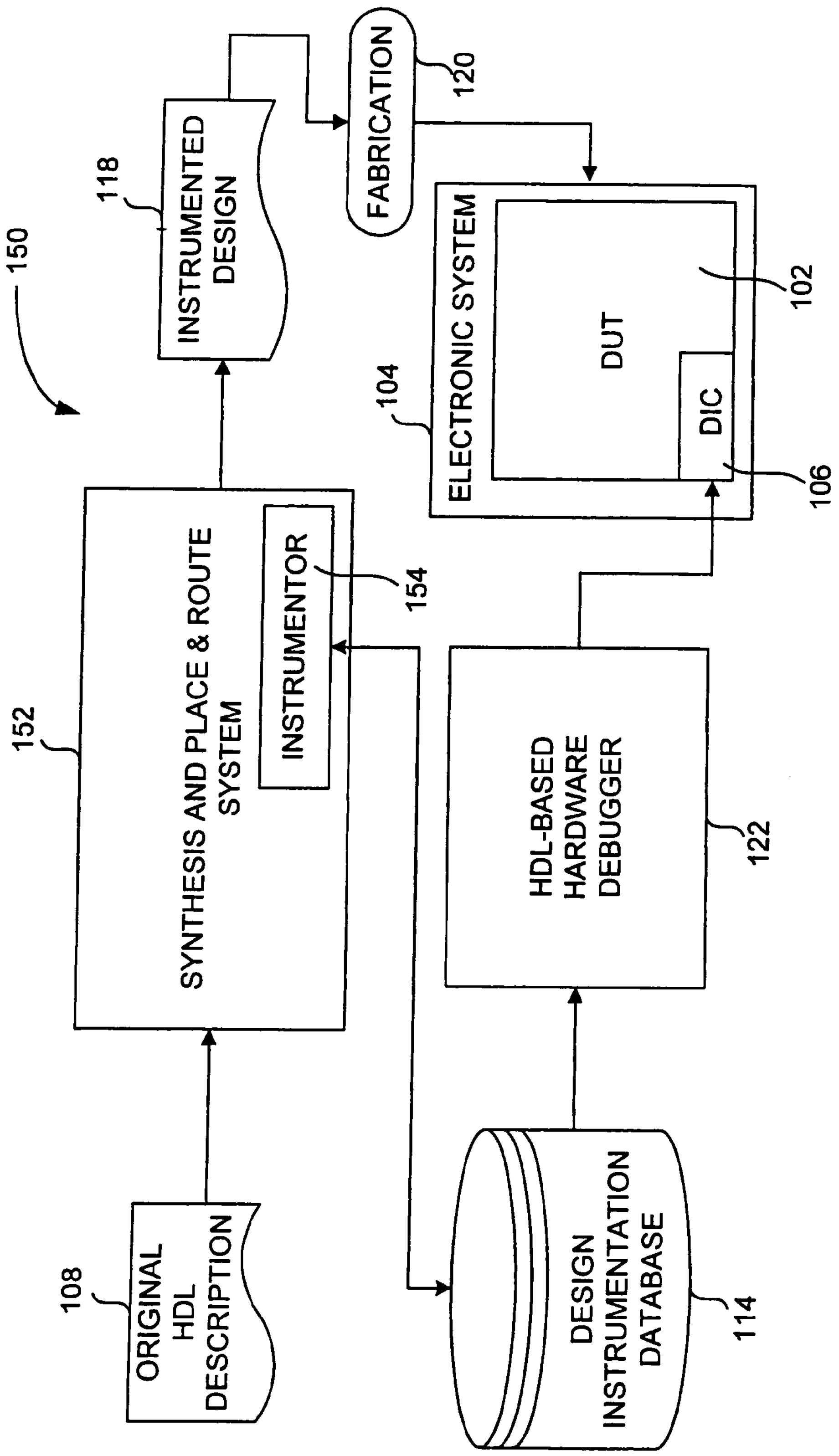

100
108
ORIGINAL HDL DESCRIPTION
110
INSTRUMENTOR
112
INSTRUMENTED HDL DESCRIPTION
116
SYNTHESIS AND PLACE & ROUTE SYSTEM
118
INSTRUMENTED DESIGN
FABRICATION
120
114
DESIGN INSTRUMENTATION DATABASE
122
HDL-BASED HARDWARE DEBUGGER
104
ELECTRONIC SYSTEM
DUT
102
DIC
106

FIG. 1A



150
ORIGINAL HDL DESCRIPTION
108
SYNTHESIS AND PLACE & ROUTE SYSTEM
152
INSTRUMENTOR
154
INSTRUMENTED DESIGN
118
FABRICATION
120
ELECTRONIC SYSTEM
104
DUT
102
DIC
106
HDL-BASED HARDWARE DEBUGGER
122
DESIGN INSTRUMENTATION DATABASE
114

FIG. 1B

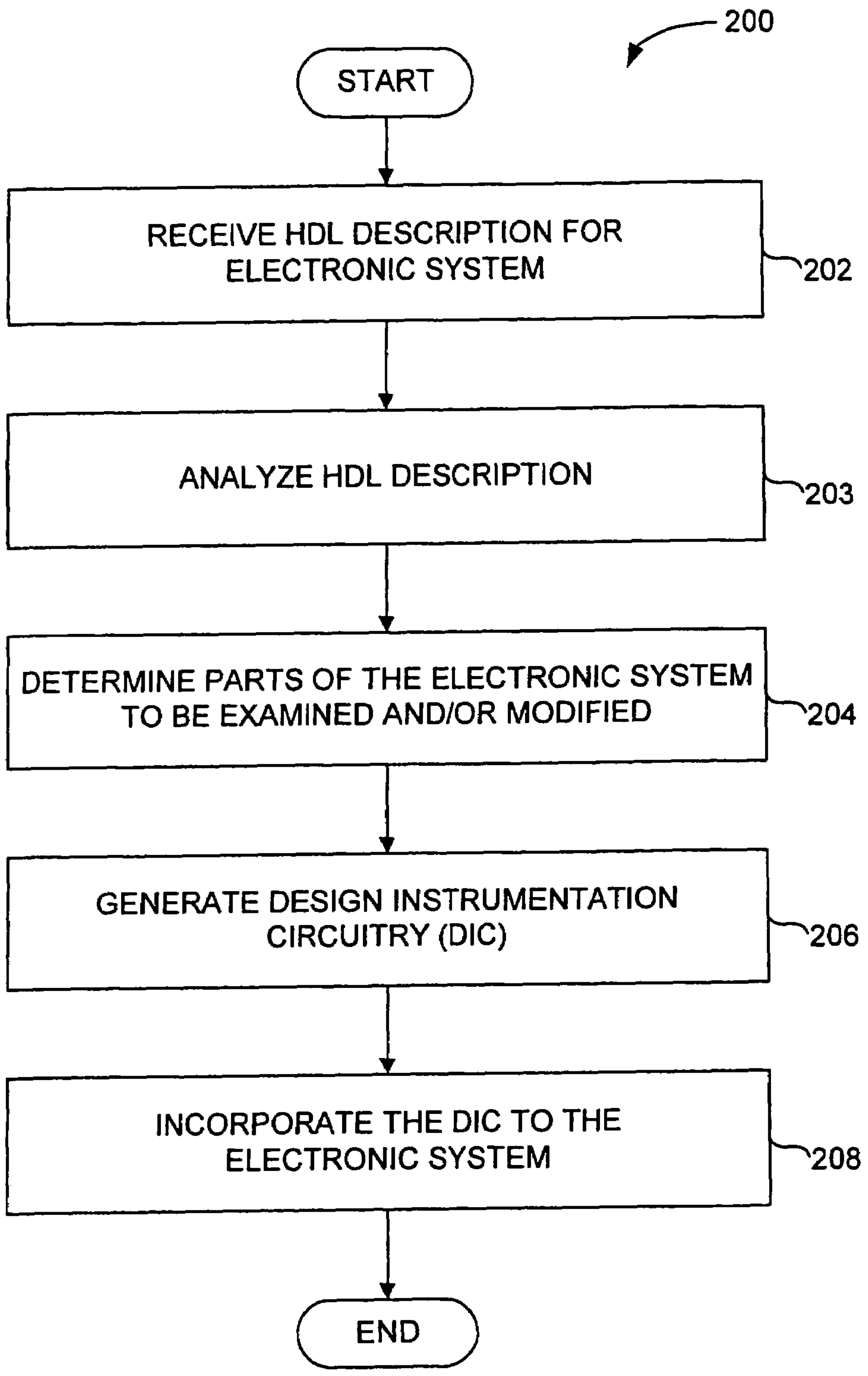

200
START
RECEIVE HDL DESCRIPTION FOR ELECTRONIC SYSTEM
202
ANALYZE HDL DESCRIPTION
203
DETERMINE PARTS OF THE ELECTRONIC SYSTEM TO BE EXAMINED AND/OR MODIFIED
204
GENERATE DESIGN INSTRUMENTATION CIRCUITRY (DIC)
206
INCORPORATE THE DIC TO THE ELECTRONIC SYSTEM
208
END

FIG. 2

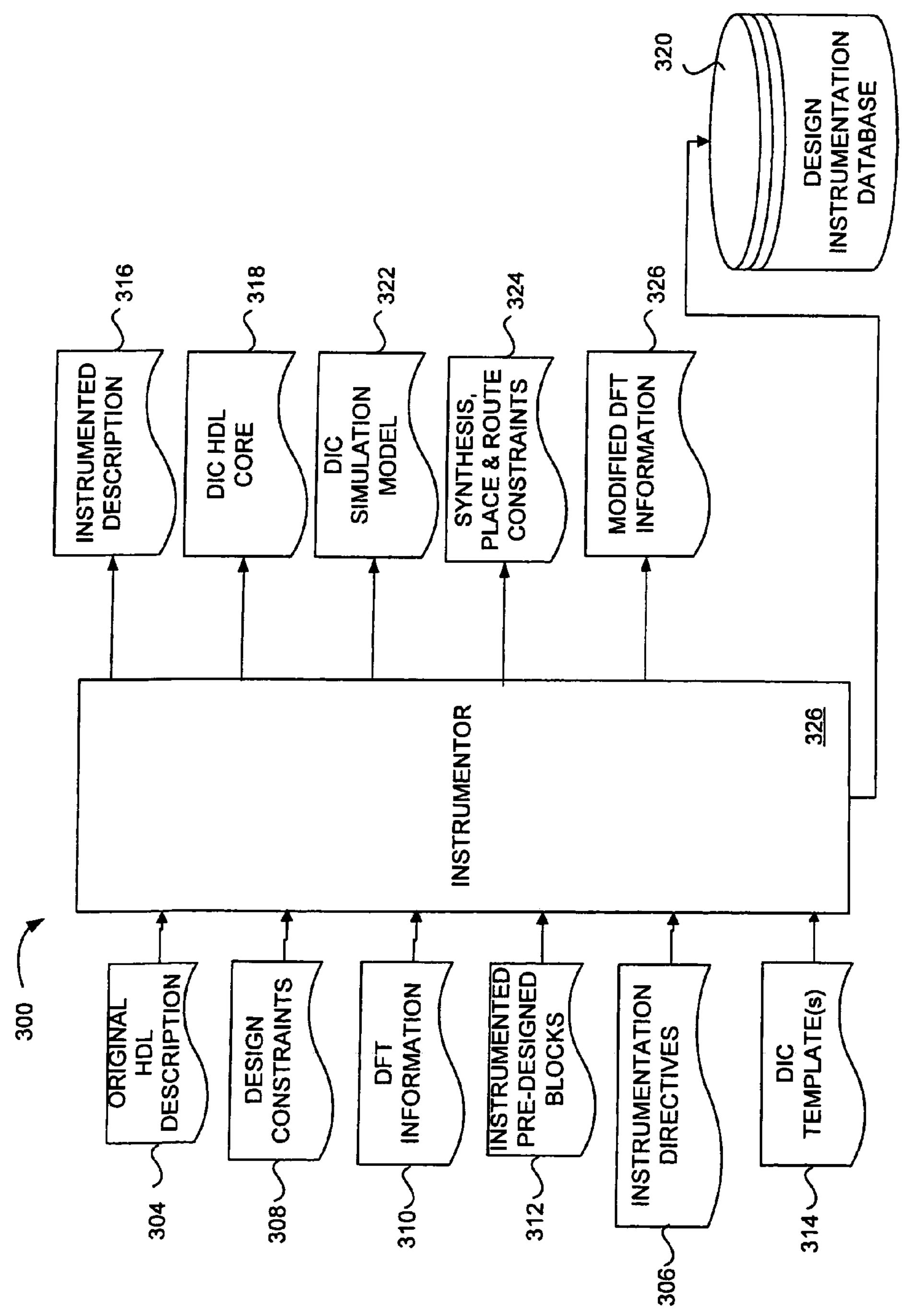

300
ORIGINAL HDL DESCRIPTION
304
DESIGN CONSTRAINTS
308
DFT INFORMATION
310
INSTRUMENTED PRE-DESIGNED BLOCKS
312
INSTRUMENTATION DIRECTIVES
306
DIC TEMPLATE(s)
314
INSTRUMENTOR
326
INSTRUMENTED DESCRIPTION
316
DIC HDL CORE
318
DIC SIMULATION MODEL
322
SYNTHESIS, PLACE & ROUTE CONSTRAINTS
324
MODIFIED DFT INFORMATION
326
DESIGN INSTRUMENTATION DATABASE
320

FIG. 3

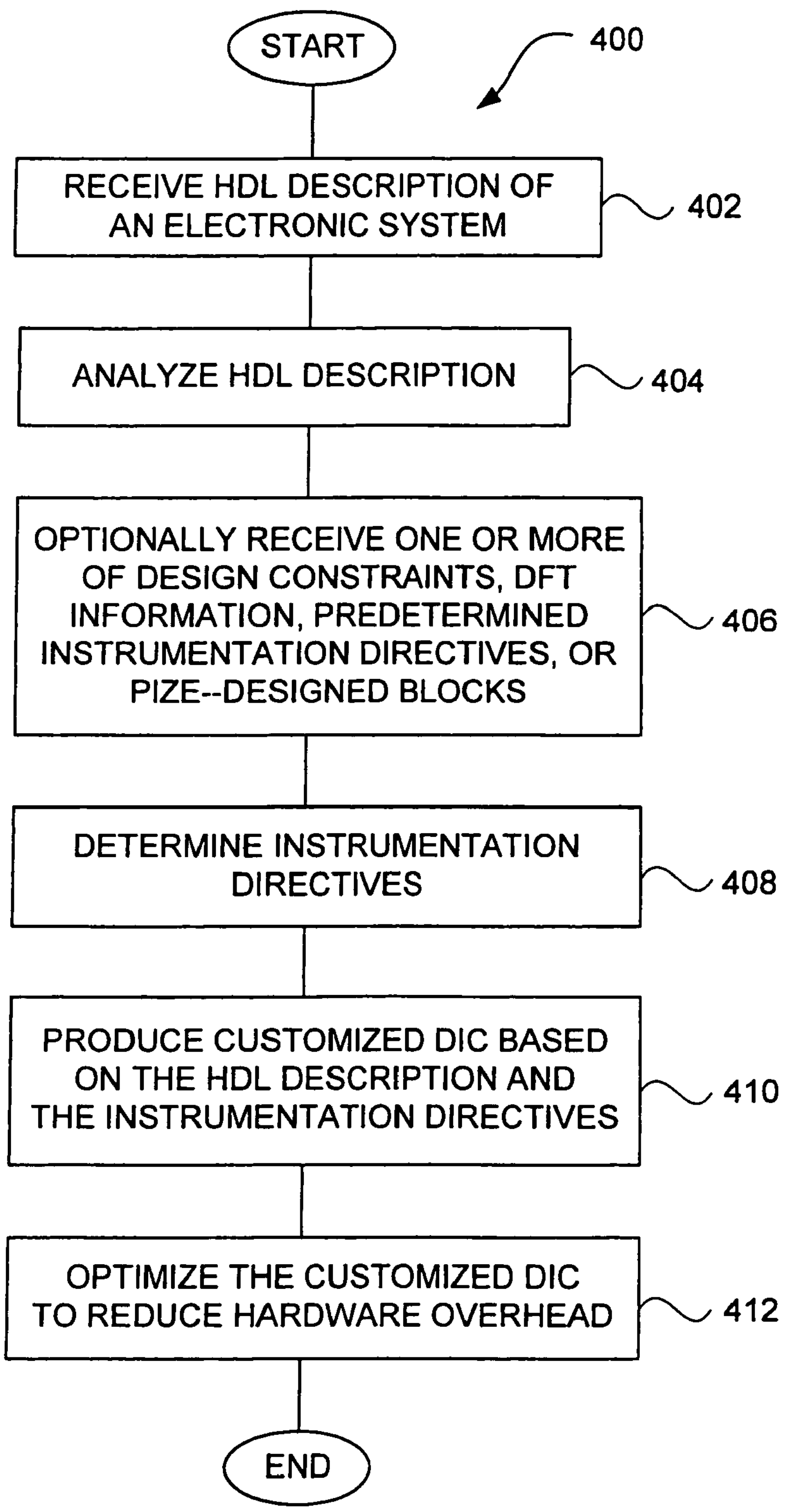

START
400
RECEIVE HDL DESCRIPTION OF AN ELECTRONIC SYSTEM
402
ANALYZE HDL DESCRIPTION
404
OPTIONALLY RECEIVE ONE OR MORE OF DESIGN CONSTRAINTS, DFT INFORMATION, PREDETERMINED INSTRUMENTATION DIRECTIVES, OR PIZE--DESIGNED BLOCKS
406
DETERMINE INSTRUMENTATION DIRECTIVES
408
PRODUCE CUSTOMIZED DIC BASED ON THE HDL DESCRIPTION AND THE INSTRUMENTATION DIRECTIVES
410
OPTIMIZE THE CUSTOMIZED DIC TO REDUCE HARDWARE OVERHEAD
412
END

FIG. 4A

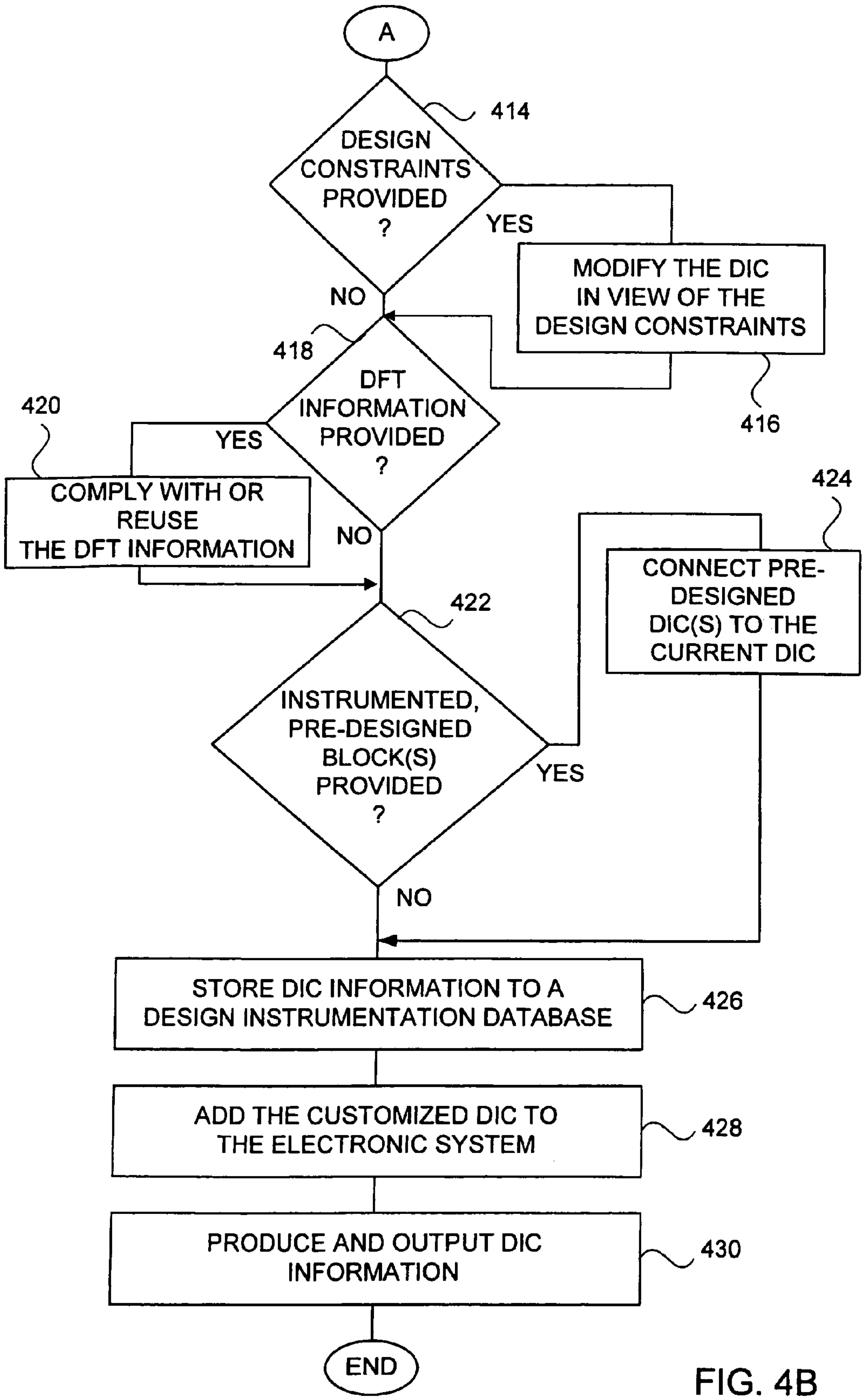

A
414
DESIGN CONSTRAINTS PROVIDED ?
YES
MODIFY THE DIC IN VIEW OF THE DESIGN CONSTRAINTS
416
NO
418
DFT INFORMATION PROVIDED ?
420
YES
COMPLY WITH OR REUSE THE DFT INFORMATION
NO
424
CONNECT PRE-DESIGNED DIC(S) TO THE CURRENT DIC
422
INSTRUMENTED, PRE-DESIGNED BLOCK(S) PROVIDED ?
YES
NO
STORE DIC INFORMATION TO A DESIGN INSTRUMENTATION DATABASE
426
ADD THE CUSTOMIZED DIC TO THE ELECTRONIC SYSTEM
428
PRODUCE AND OUTPUT DIC INFORMATION
430
END

FIG. 4B

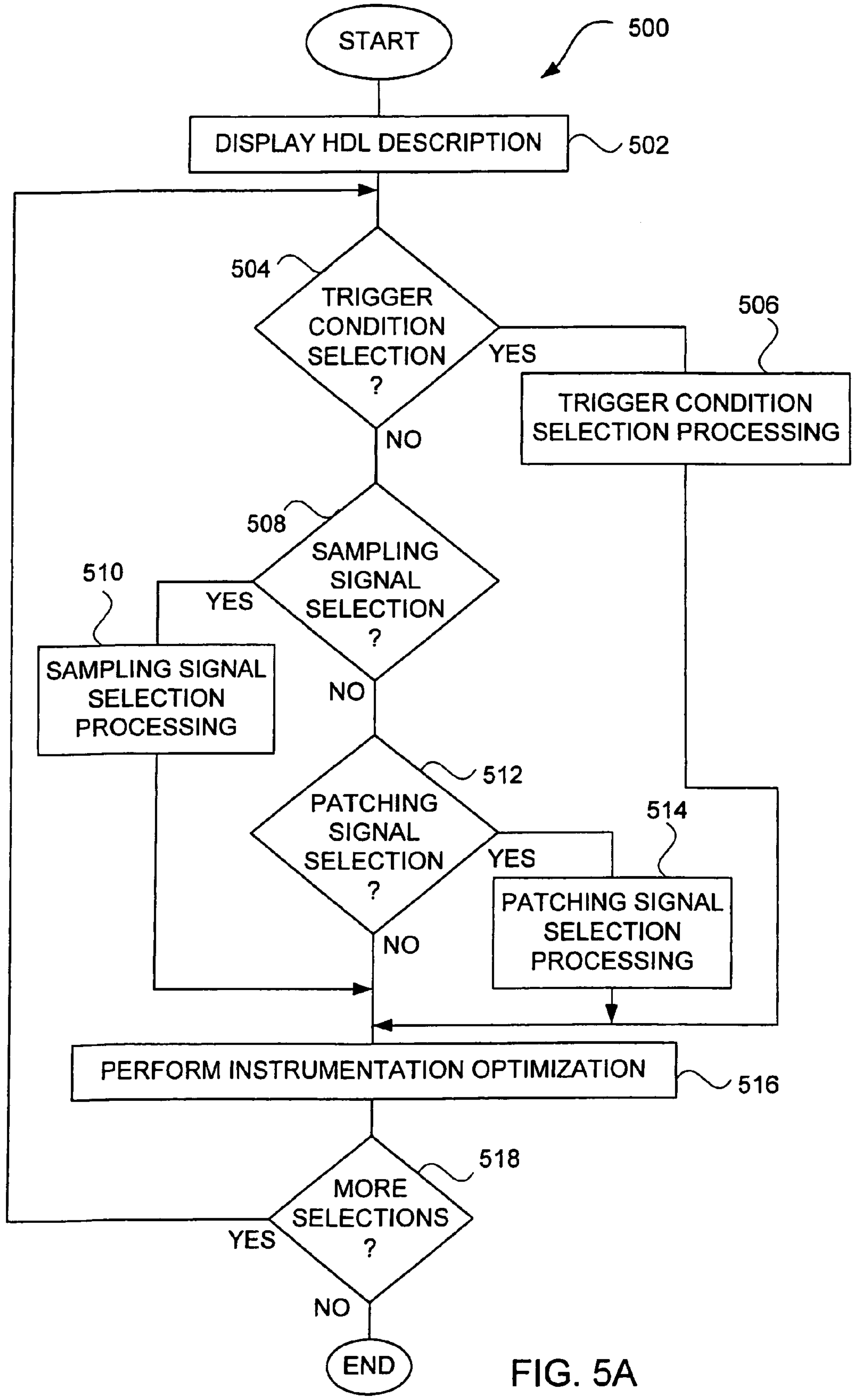
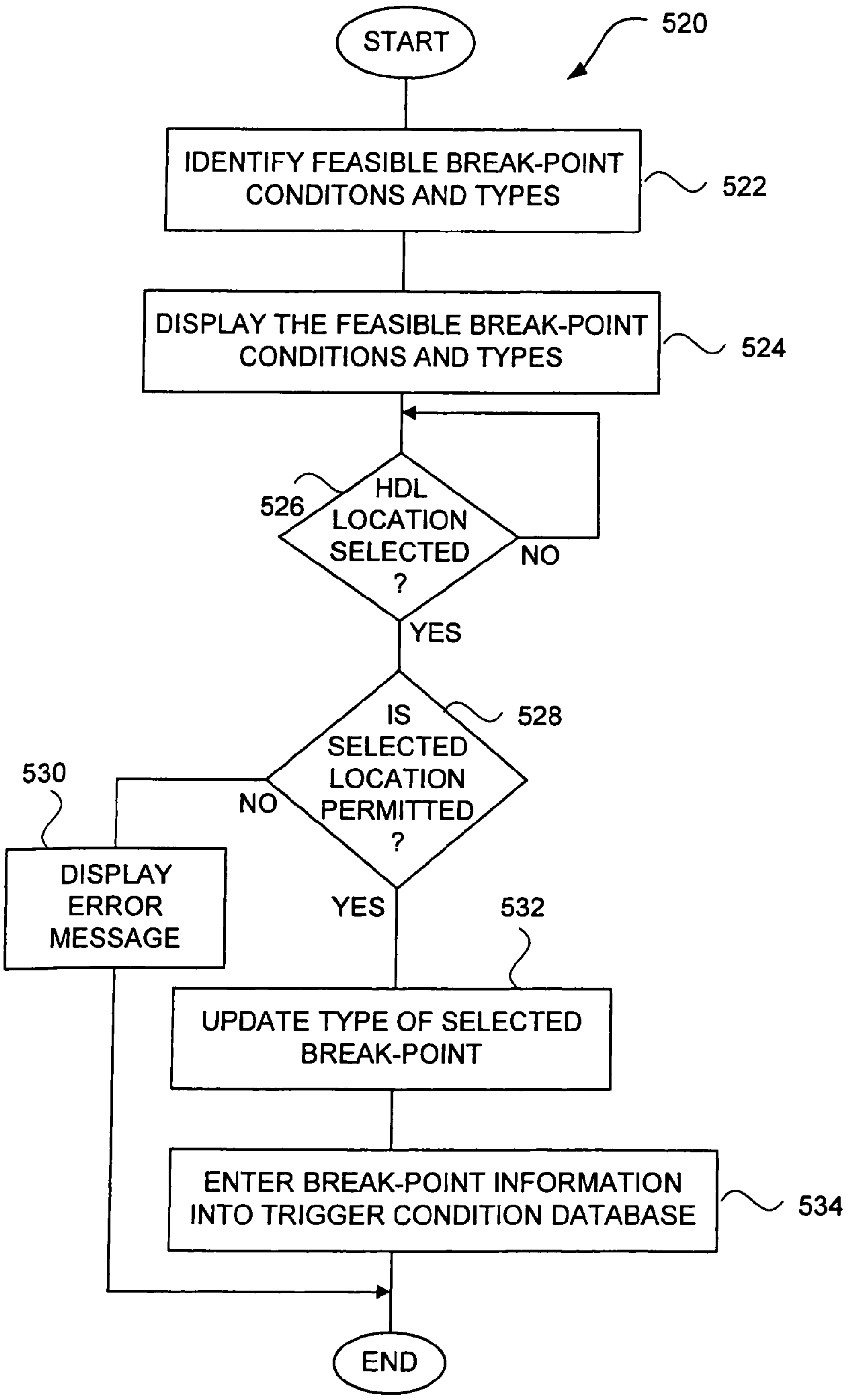

START
500
DISPLAY HDL DESCRIPTION
502
504
TRIGGER CONDITION SELECTION ?
YES
NO
506
TRIGGER CONDITION SELECTION PROCESSING
508
SAMPLING SIGNAL SELECTION ?
YES
NO
510
SAMPLING SIGNAL SELECTION PROCESSING
512
PATCHING SIGNAL SELECTION ?
YES
NO
514
PATCHING SIGNAL SELECTION PROCESSING
PERFORM INSTRUMENTATION OPTIMIZATION
516
518
MORE SELECTIONS ?
YES
NO
END

FIG. 5A



START
520
IDENTIFY FEASIBLE BREAK-POINT CONDITONS AND TYPES
522
DISPLAY THE FEASIBLE BREAK-POINT CONDITIONS AND TYPES
524
526
HDL LOCATION SELECTED ?
NO
YES
IS SELECTED LOCATION PERMITTED ?
528
530
NO
DISPLAY ERROR MESSAGE
YES
532
UPDATE TYPE OF SELECTED BREAK-POINT
ENTER BREAK-POINT INFORMATION INTO TRIGGER CONDITION DATABASE
534
END

FIG. 5B

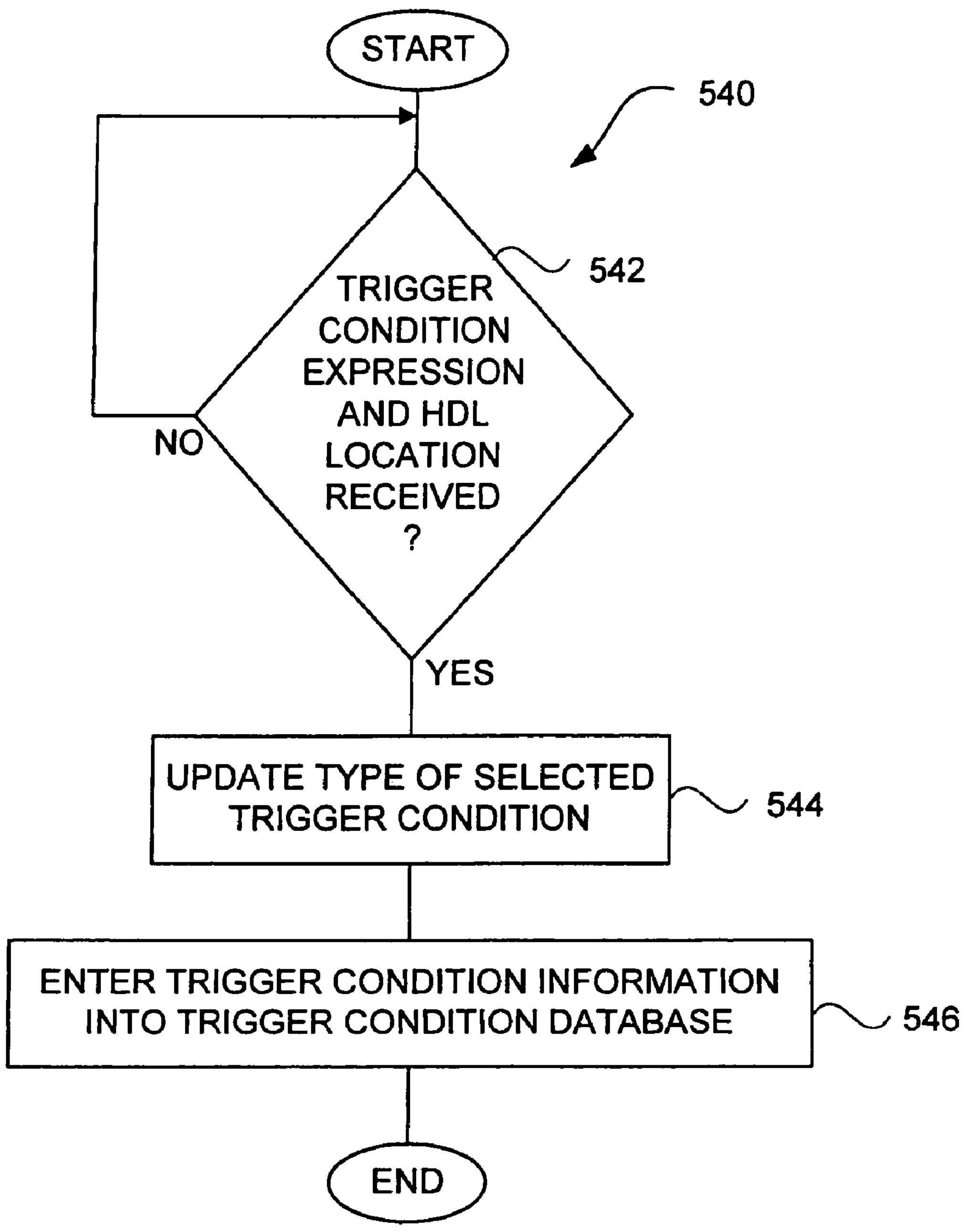

START
540
TRIGGER CONDITION EXPRESSION AND HDL LOCATION RECEIVED ?
542
NO
YES
UPDATE TYPE OF SELECTED TRIGGER CONDITION
544
ENTER TRIGGER CONDITION INFORMATION INTO TRIGGER CONDITION DATABASE
546
END

FIG. 5C

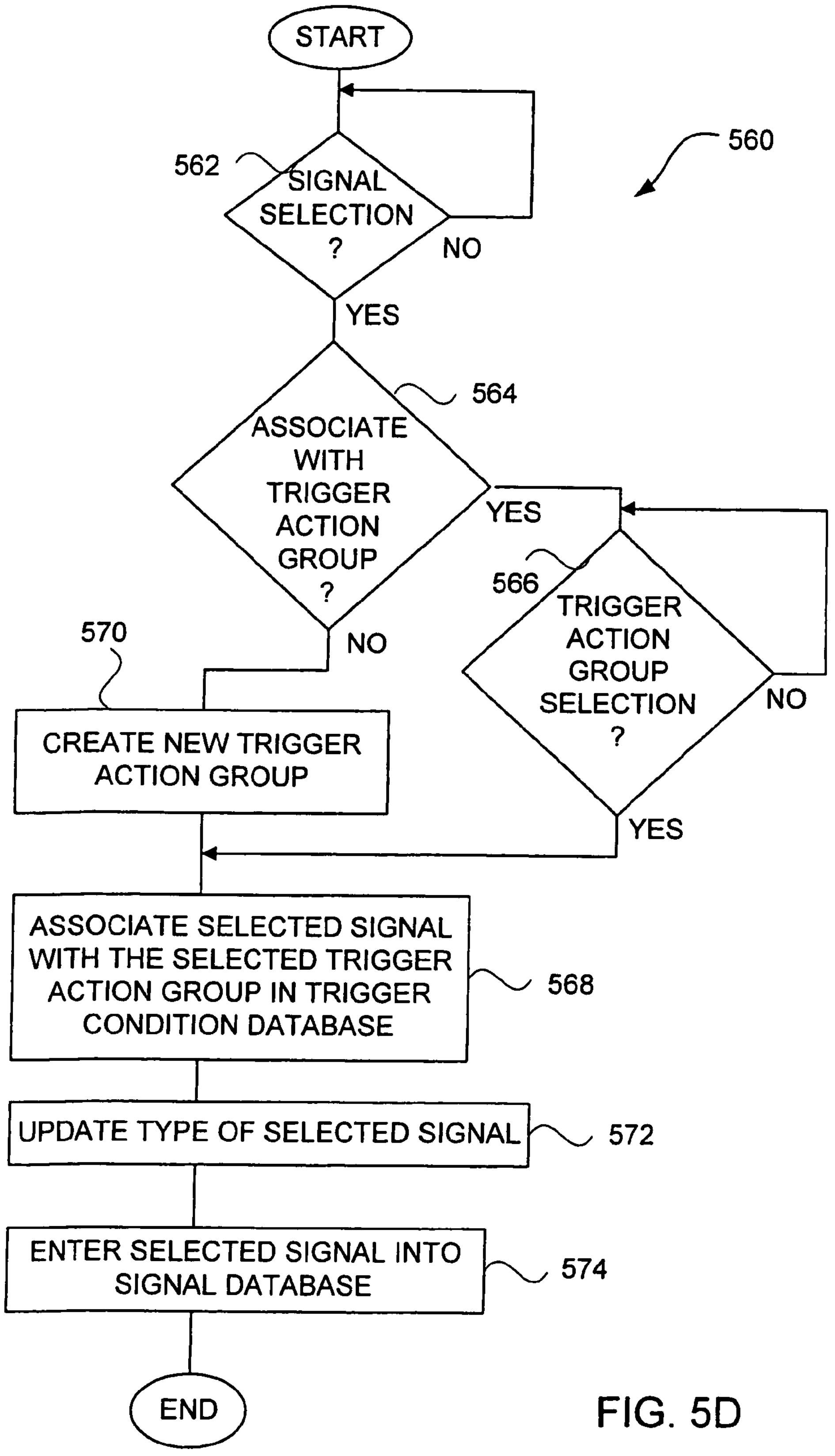

START
562
SIGNAL SELECTION ?
NO
YES
560
564
ASSOCIATE WITH TRIGGER ACTION GROUP ?
YES
NO
566
TRIGGER ACTION GROUP SELECTION ?
NO
YES
570
CREATE NEW TRIGGER ACTION GROUP
ASSOCIATE SELECTED SIGNAL WITH THE SELECTED TRIGGER ACTION GROUP IN TRIGGER CONDITION DATABASE
568
UPDATE TYPE OF SELECTED SIGNAL
572
ENTER SELECTED SIGNAL INTO SIGNAL DATABASE
574
END

FIG. 5D

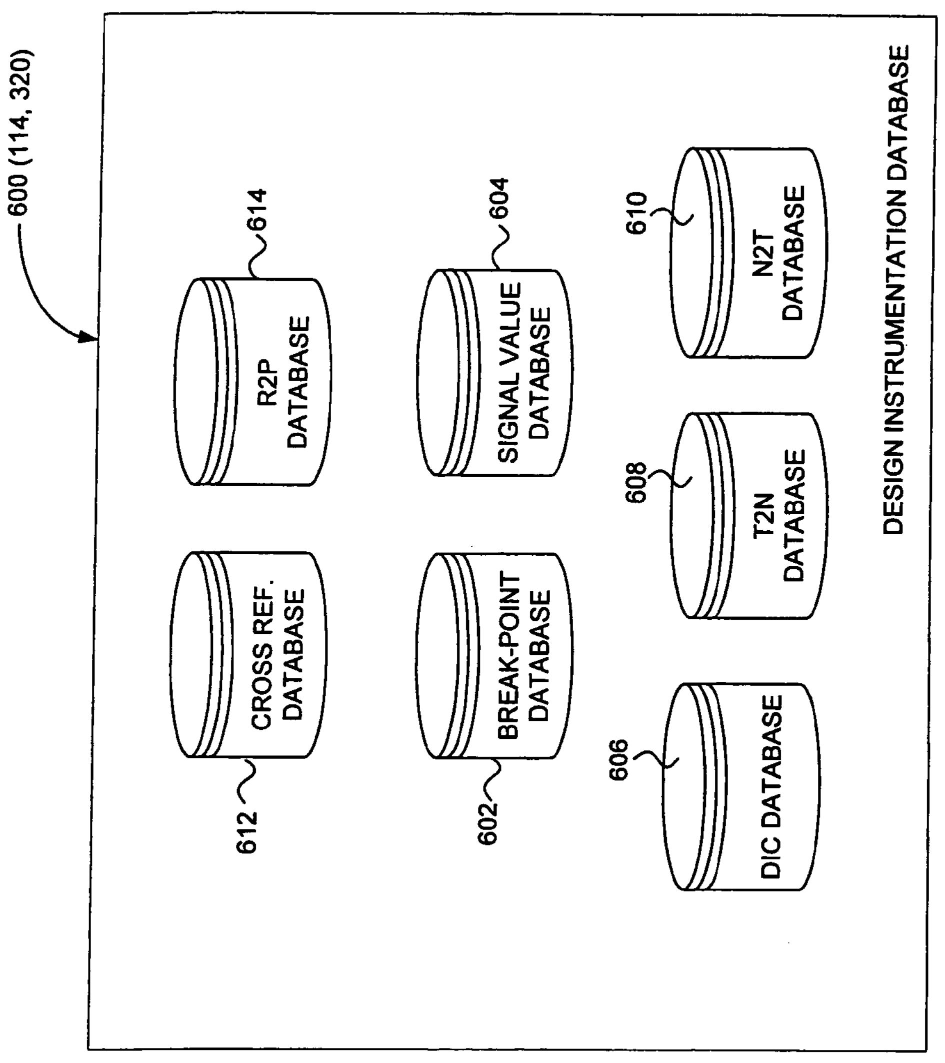

600 (114, 320)
DESIGN INSTRUMENTATION DATABASE
R2P DATABASE
614
CROSS REF. DATABASE
612
SIGNAL VALUE DATABASE
604
BREAK-POINT DATABASE
602
N2T DATABASE
610
T2N DATABASE
608
DIC DATABASE
606

FIG. 6

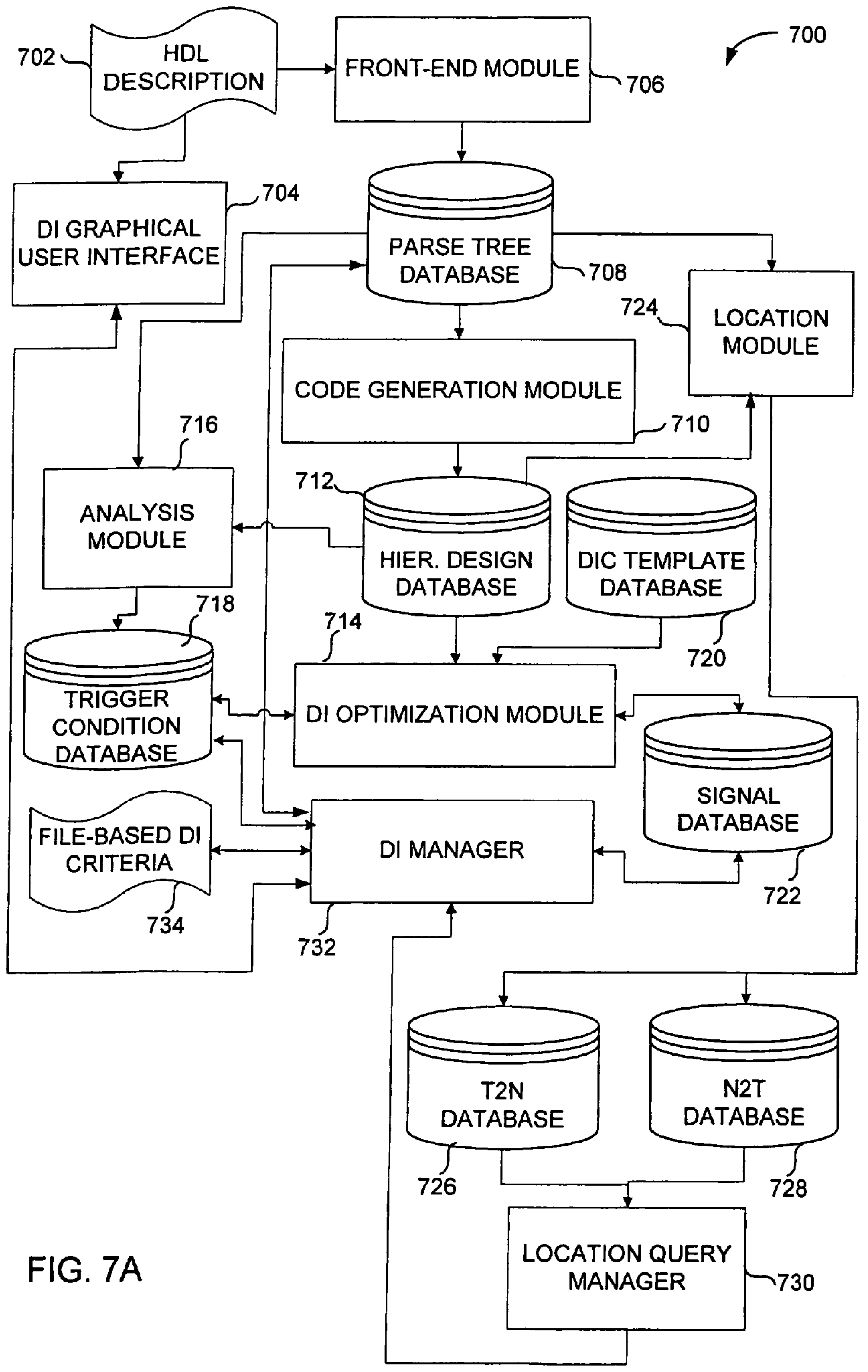

700
702
HDL DESCRIPTION
FRONT-END MODULE
706
DI GRAPHICAL USER INTERFACE
704
PARSE TREE DATABASE
708
724
LOCATION MODULE
CODE GENERATION MODULE
710
716
ANALYSIS MODULE
712
HIER. DESIGN DATABASE
DIC TEMPLATE DATABASE
718
714
720
TRIGGER CONDITION DATABASE
DI OPTIMIZATION MODULE
SIGNAL DATABASE
FILE-BASED DI CRITERIA
DI MANAGER
722
734
732
T2N DATABASE
N2T DATABASE
726
728
LOCATION QUERY MANAGER
730

FIG. 7A

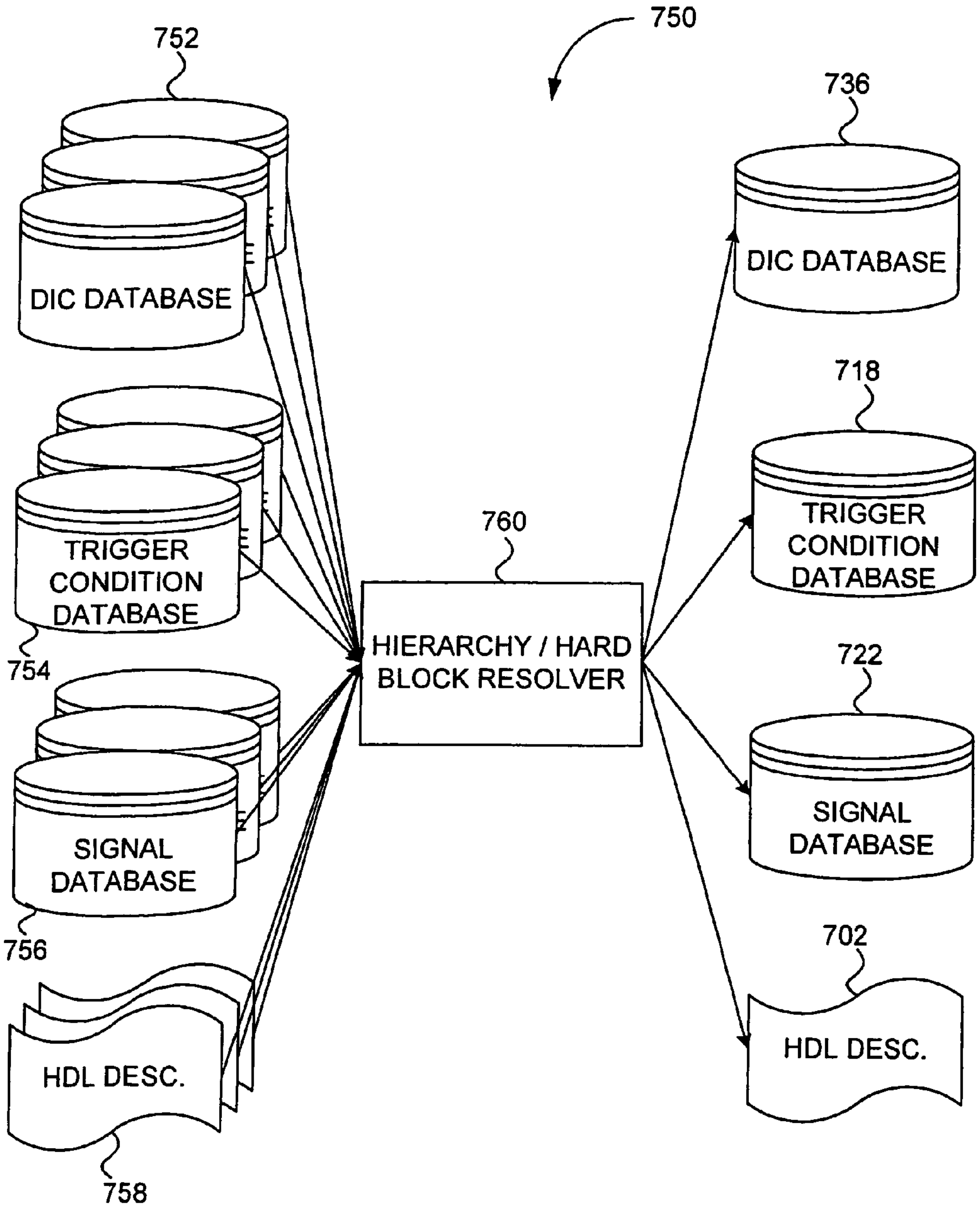

750
752
DIC DATABASE
TRIGGER CONDITION DATABASE
754
SIGNAL DATABASE
756
HDL DESC.
758
760
HIERARCHY / HARD BLOCK RESOLVER
736
DIC DATABASE
718
TRIGGER CONDITION DATABASE
722
SIGNAL DATABASE
702
HDL DESC.

FIG. 7B

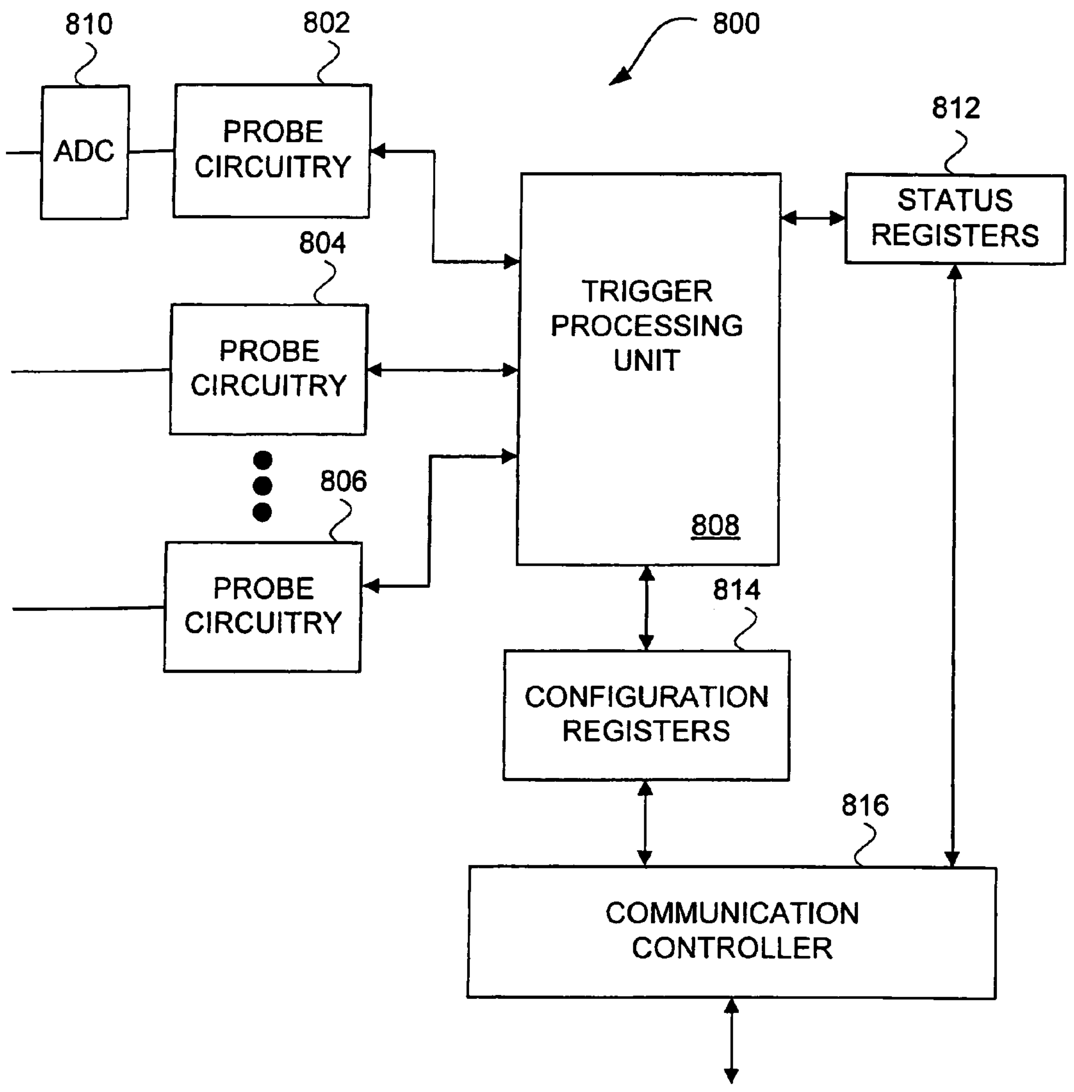
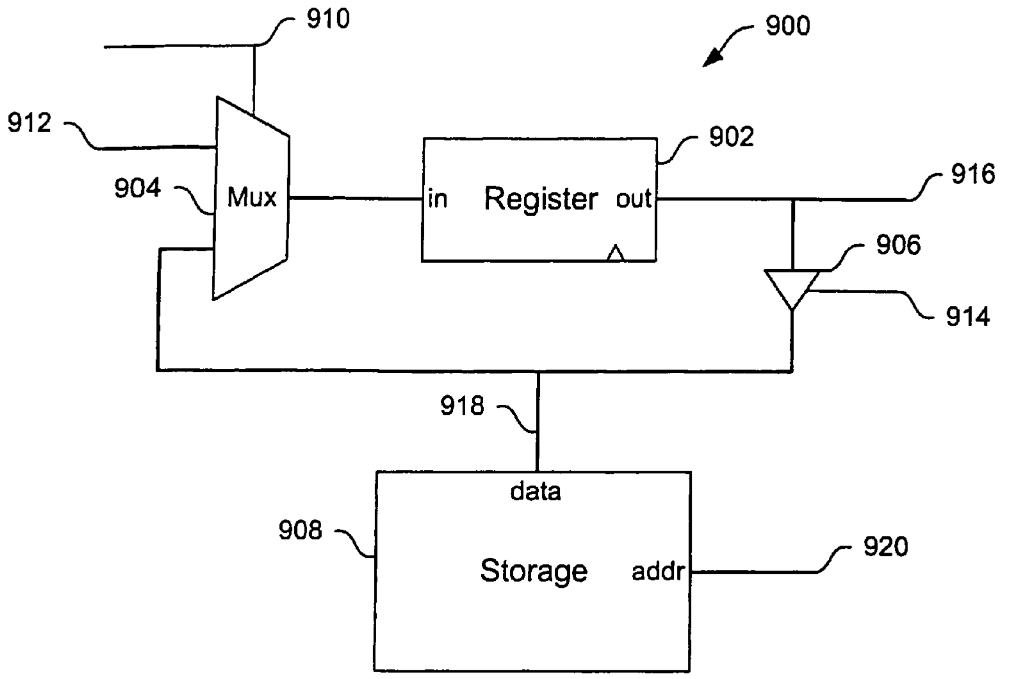
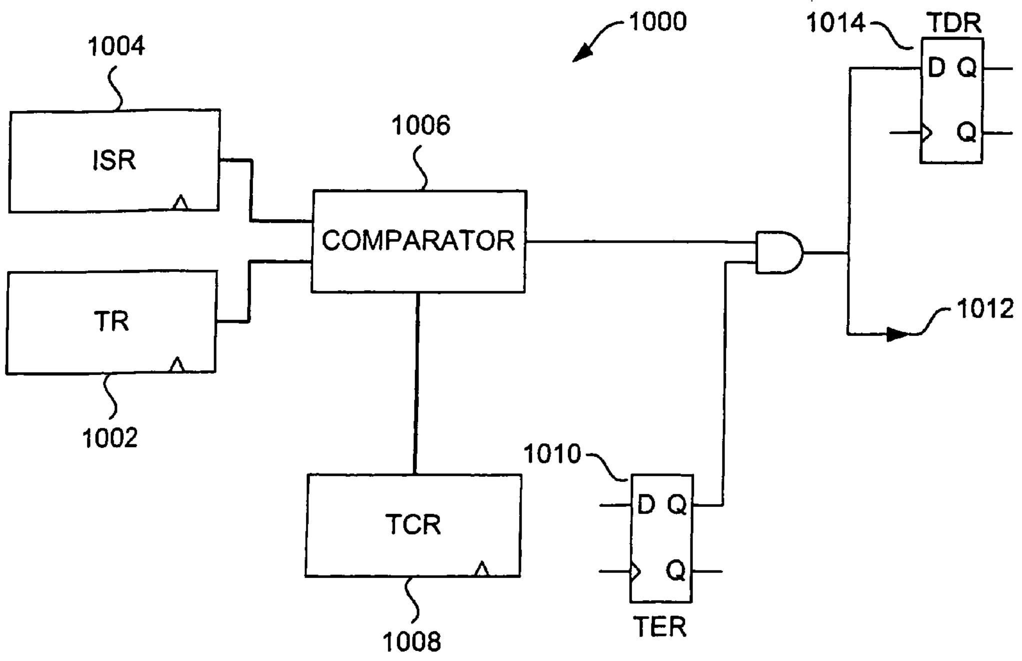

810
802
800
812
ADC
PROBE CIRCUITRY
STATUS REGISTERS
804
TRIGGER PROCESSING UNIT
PROBE CIRCUITRY
806
808
PROBE CIRCUITRY
814
CONFIGURATION REGISTERS
816
COMMUNICATION CONTROLLER

FIG. 8



910
900
912
902
904
Mux
in
Register
out
916
906
914
918
data
908
Storage
addr
920

FIG. 9



1000
1014
TDR
D Q
Q
1004
ISR
1006
COMPARATOR
TR
1012
1002
1010
D Q
Q
TCR
1008
TER

FIG. 10

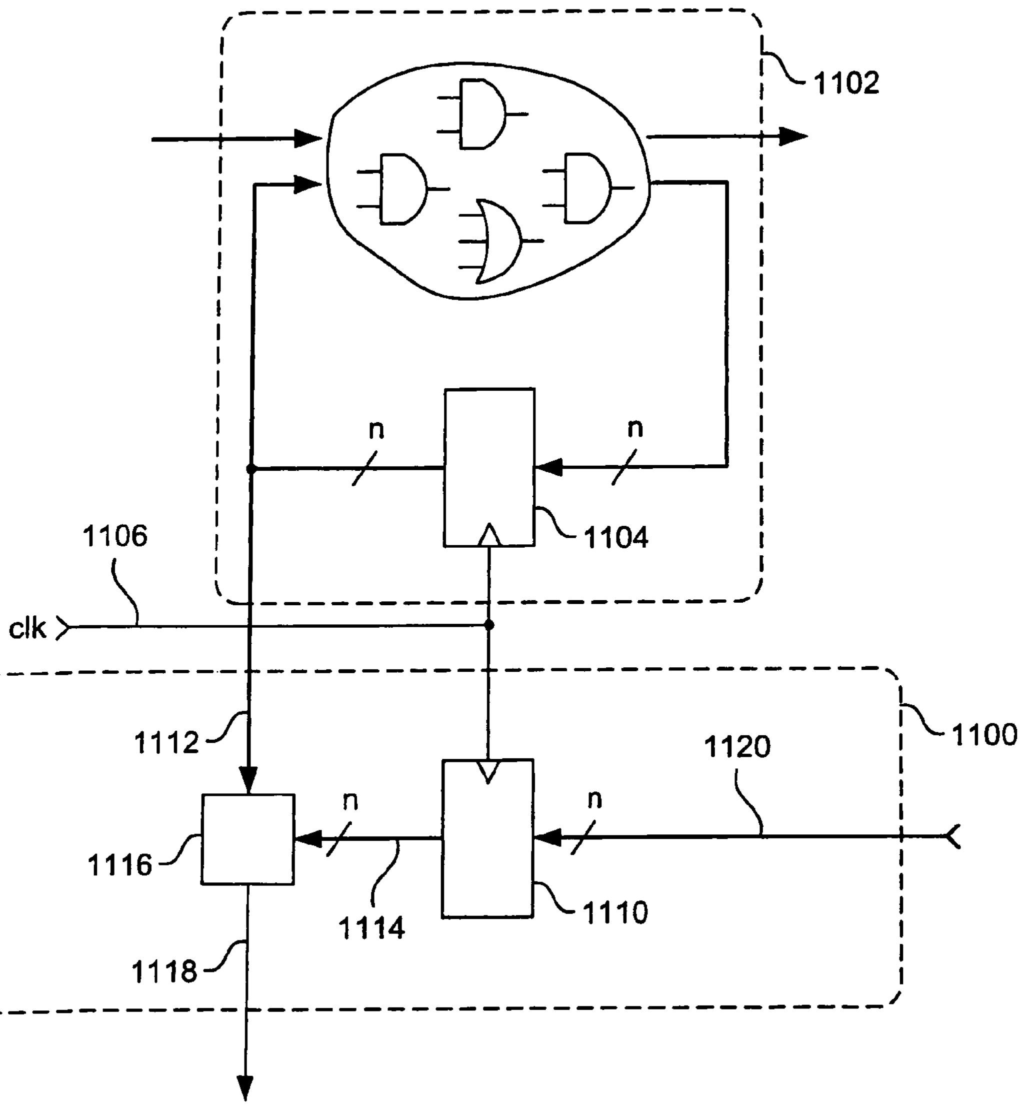
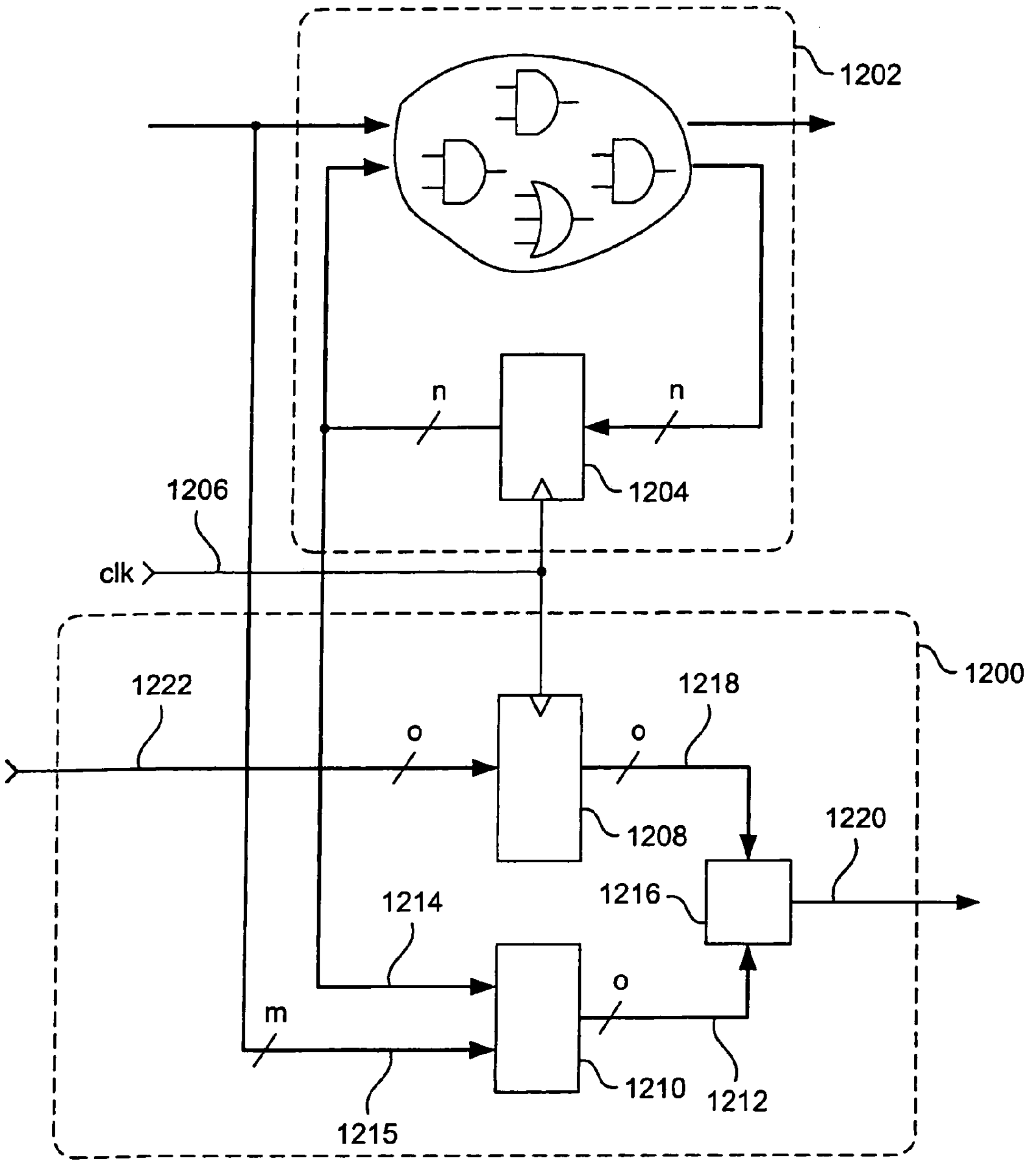


FIG. 11



1202
n
n
1204
1206
clk
1200
1222
o
1218
o
1208
1220
1216
1214
o
m
1210
1212
1215

FIG. 12

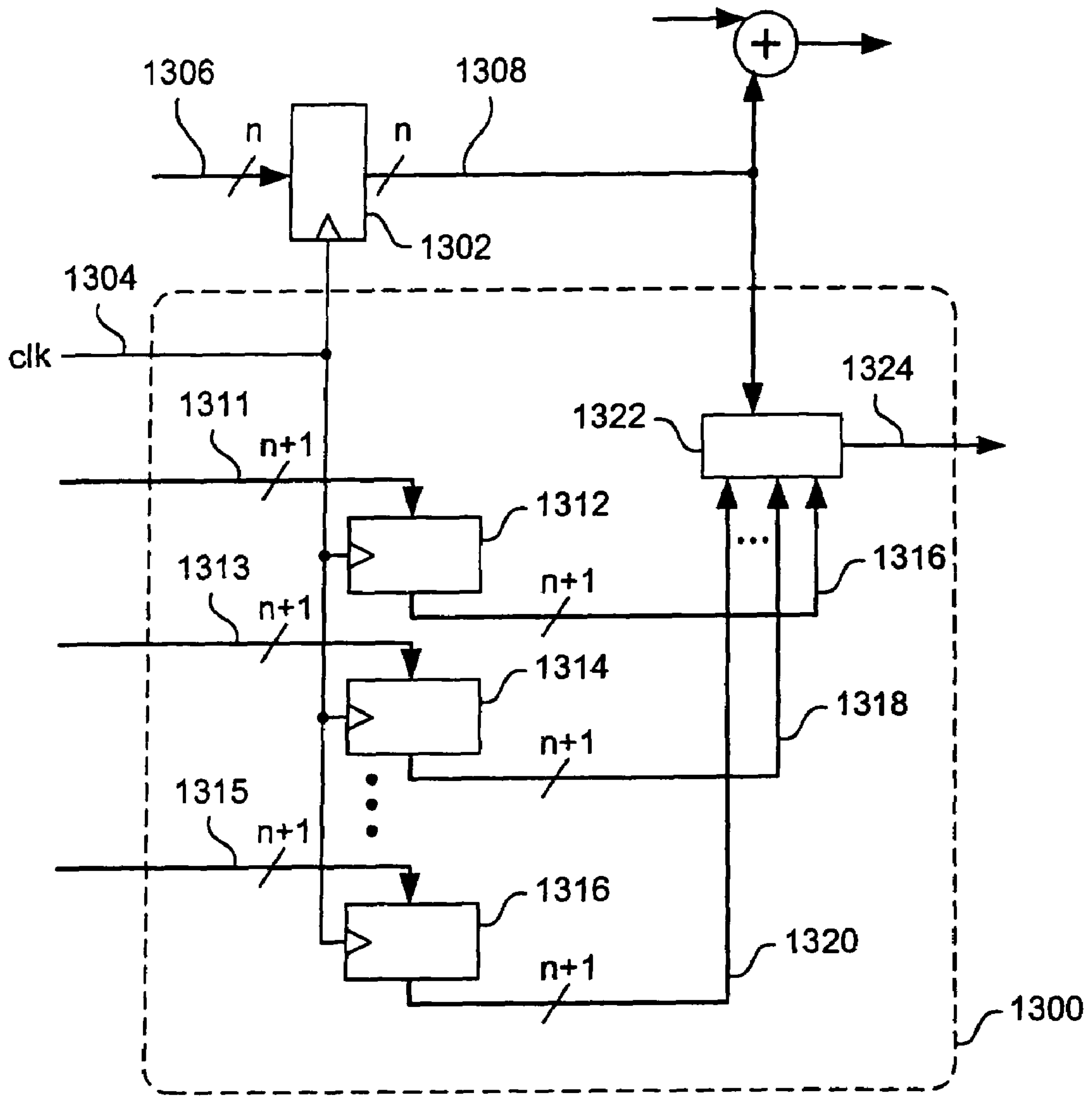

1306
n
n
1308
1302
1304
clk
1311
n+1
1312
1313
n+1
1314
1315
n+1
1316
n+1
n+1
n+1
1320
1318
1316
1322
1324
1300

FIG. 13

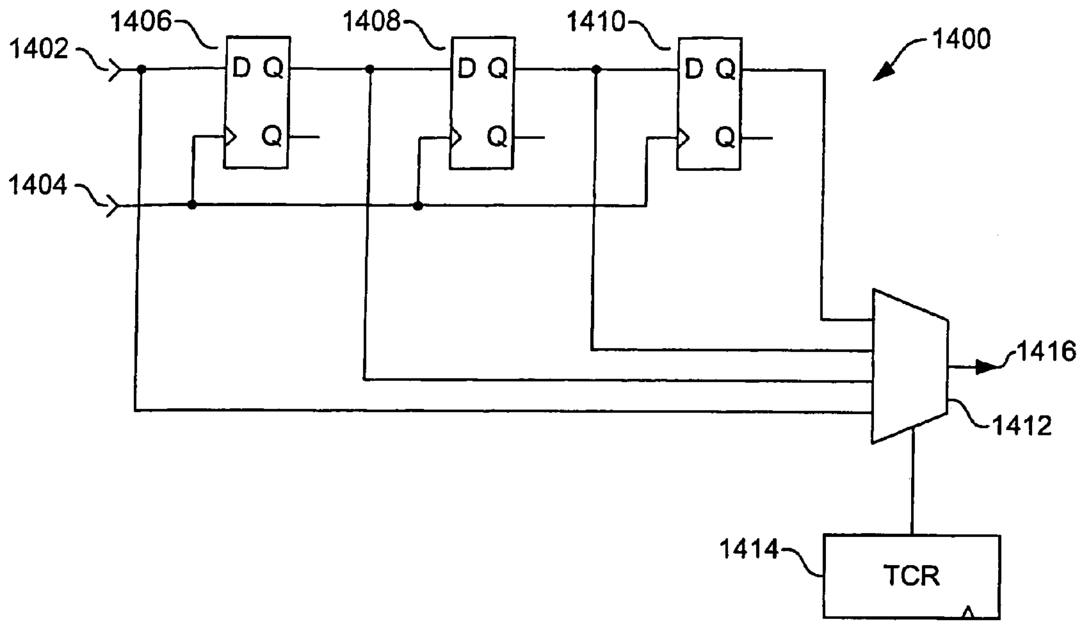


FIG. 14

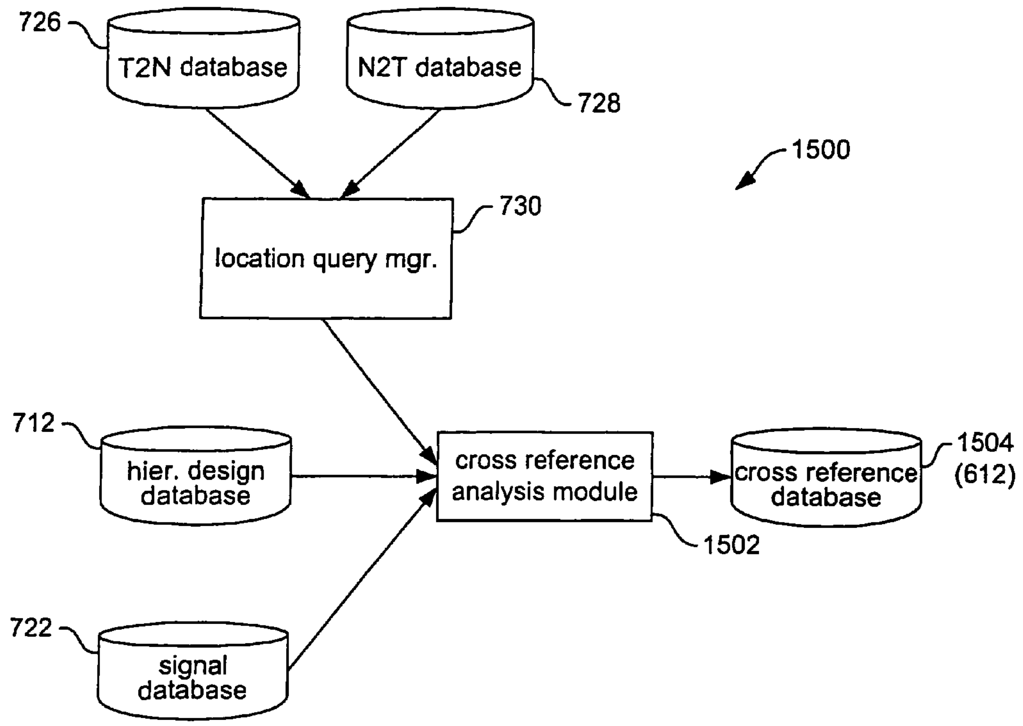

726
T2N database
N2T database
728
1500
730
location query mgr.
712
hier. design database
cross reference analysis module
cross reference database
1504
(612)
1502
722
signal database

FIG. 15

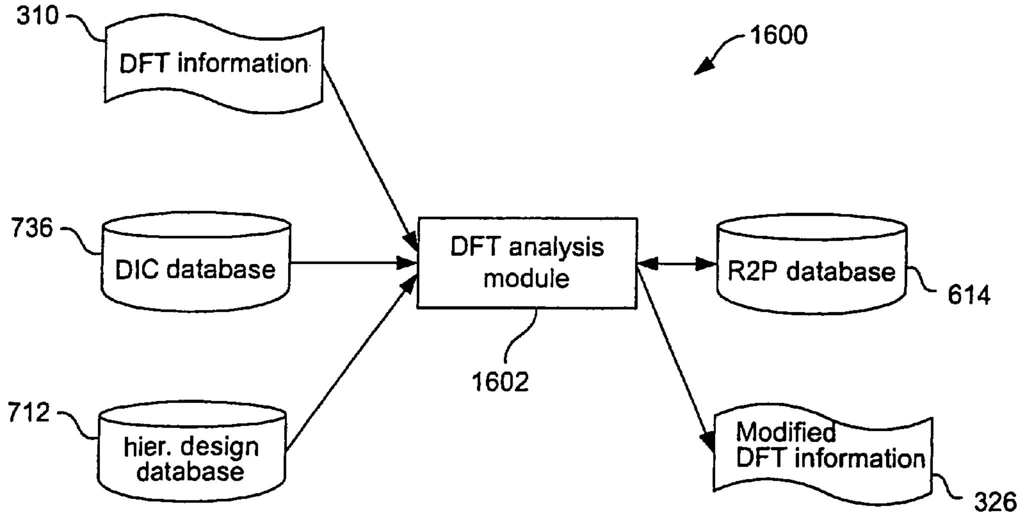

310
DFT information
1600
736
DIC database
DFT analysis module
R2P database
614
1602
712
hier. design database
Modified DFT information
326

FIG. 16

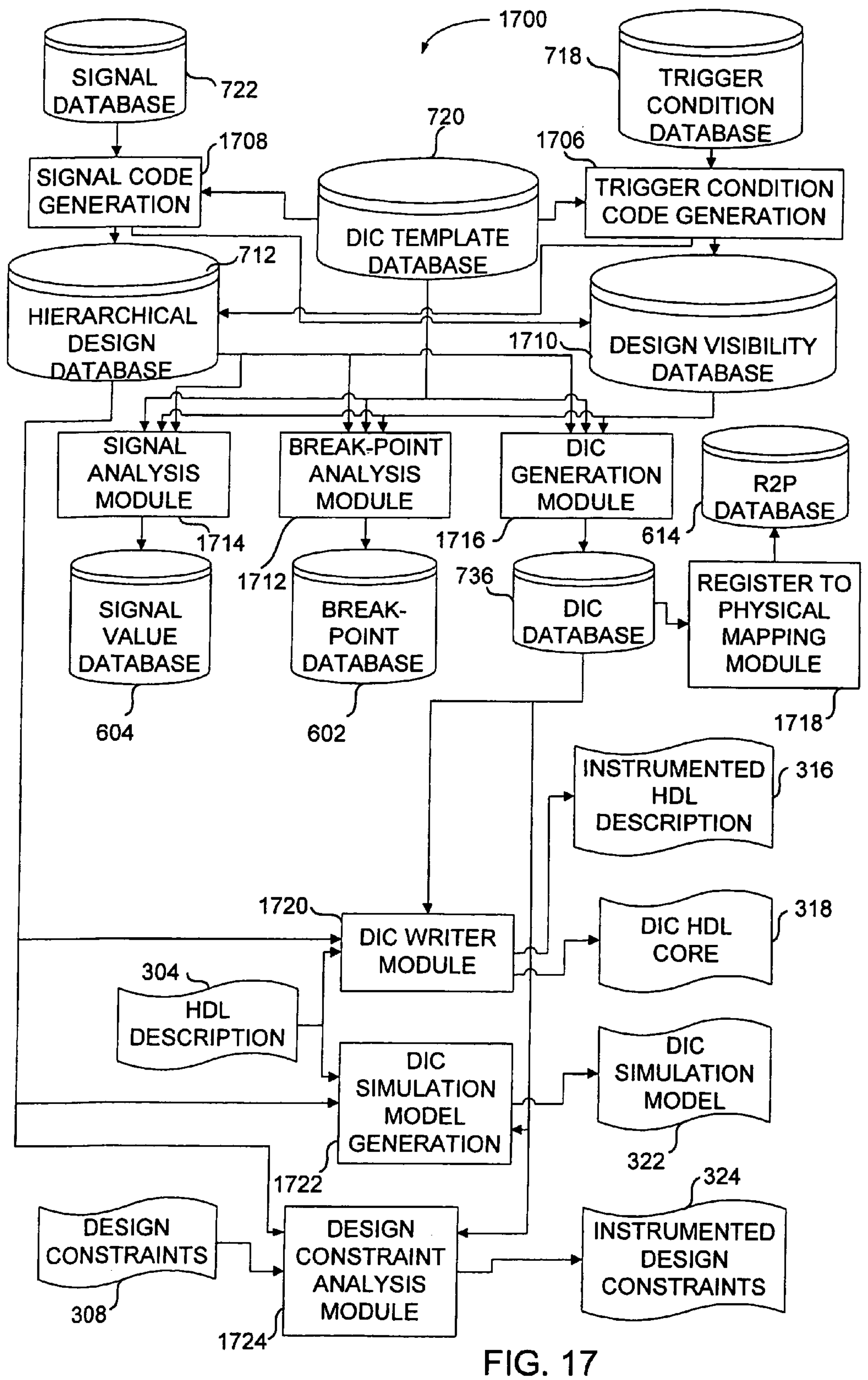

1700
SIGNAL DATABASE
722
718
TRIGGER CONDITION DATABASE
720
1708
1706
SIGNAL CODE GENERATION
DIC TEMPLATE DATABASE
TRIGGER CONDITION CODE GENERATION
712
HIERARCHICAL DESIGN DATABASE
1710
DESIGN VISIBILITY DATABASE
SIGNAL ANALYSIS MODULE
BREAK-POINT ANALYSIS MODULE
DIC GENERATION MODULE
R2P DATABASE
1714
1712
1716
614
736
SIGNAL VALUE DATABASE
BREAK-POINT DATABASE
DIC DATABASE
REGISTER TO PHYSICAL MAPPING MODULE
604
602
1718
INSTRUMENTED HDL DESCRIPTION
316
1720
DIC WRITER MODULE
DIC HDL CORE
318
304
HDL DESCRIPTION
DIC SIMULATION MODEL GENERATION
DIC SIMULATION MODEL
322
1722
324
DESIGN CONSTRAINTS
DESIGN CONSTRAINT ANALYSIS MODULE
INSTRUMENTED DESIGN CONSTRAINTS
308
1724

FIG. 17

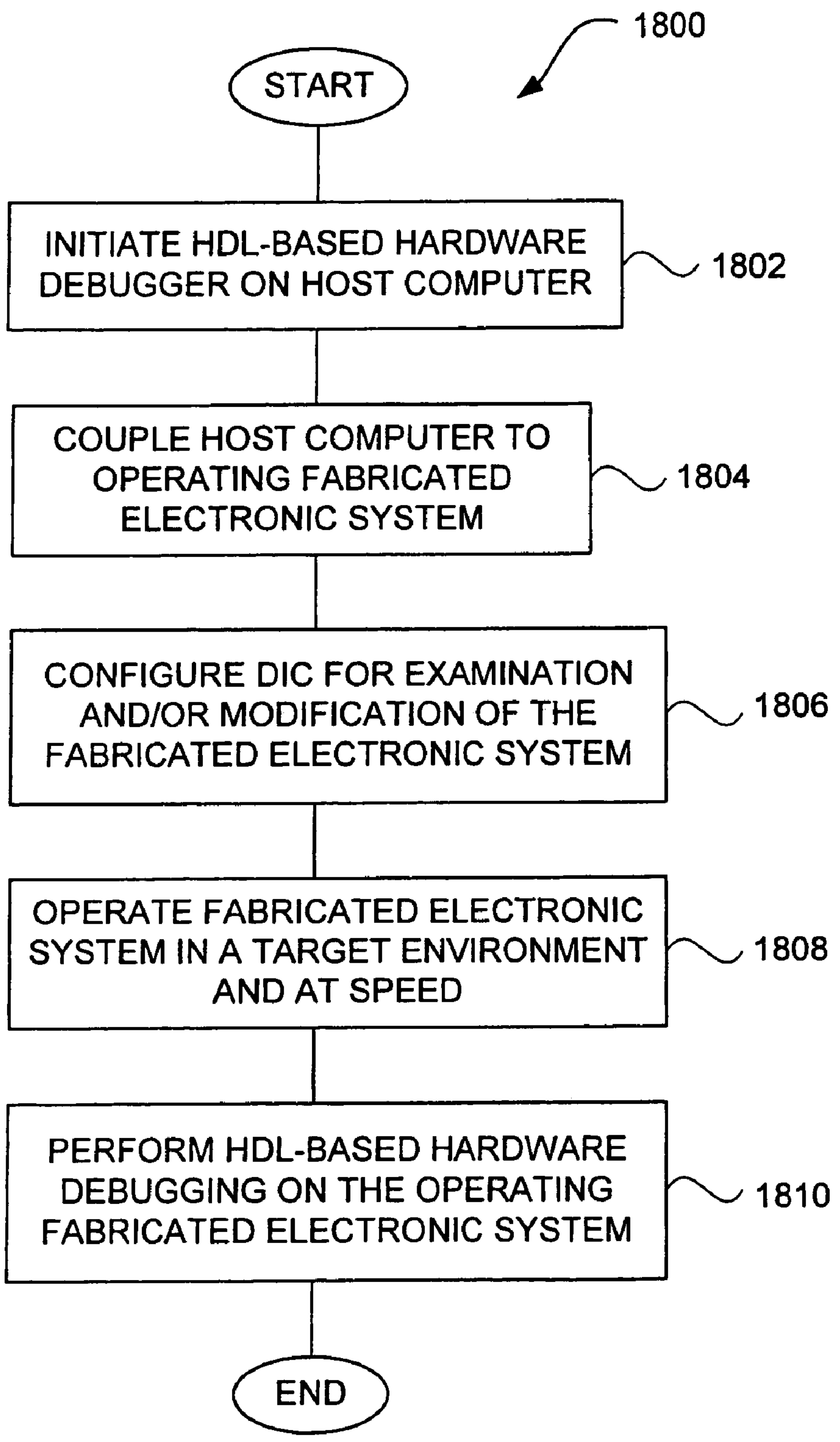
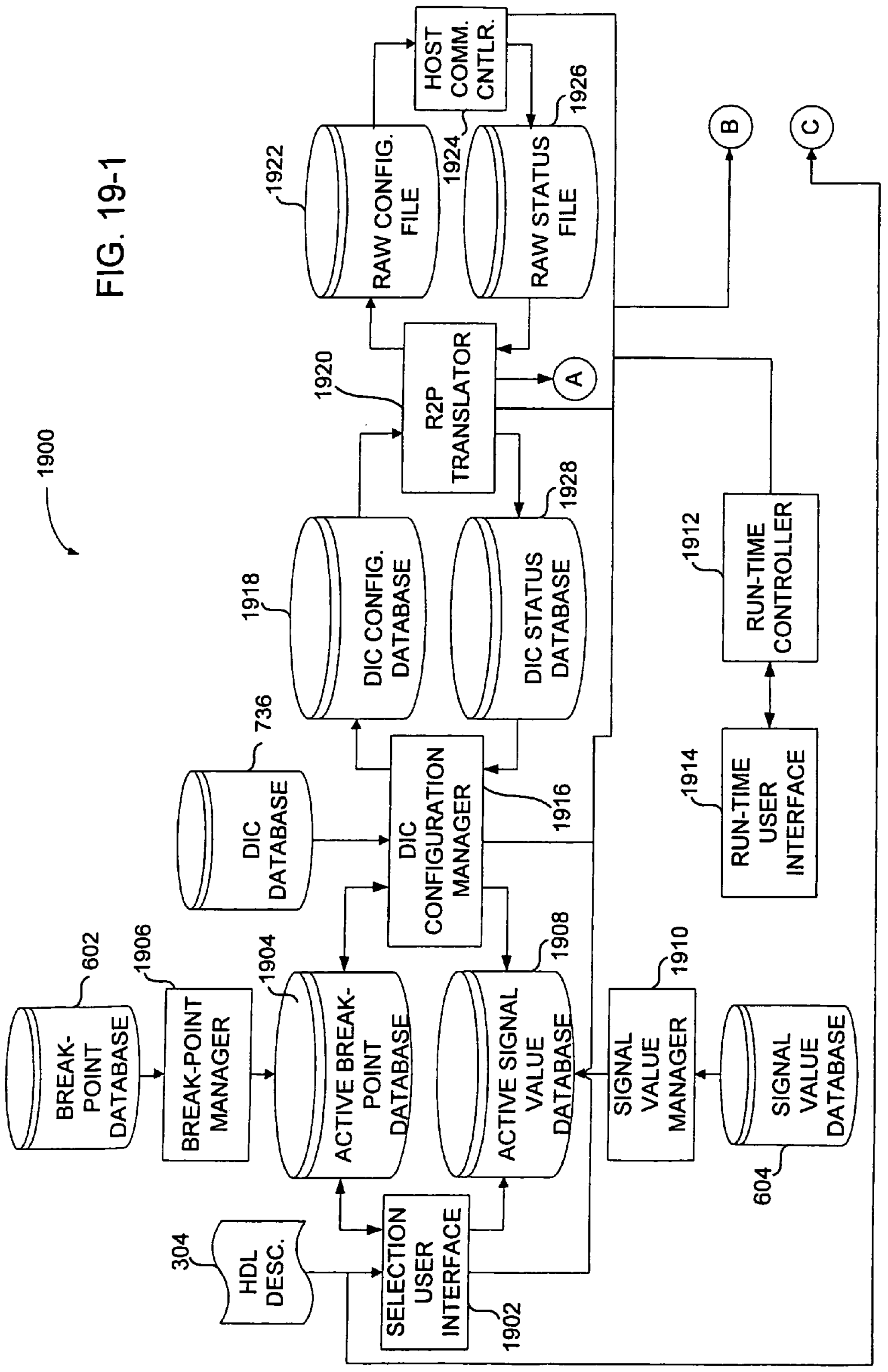

1800
START
INITIATE HDL-BASED HARDWARE DEBUGGER ON HOST COMPUTER
1802
COUPLE HOST COMPUTER TO OPERATING FABRICATED ELECTRONIC SYSTEM
1804
CONFIGURE DIC FOR EXAMINATION AND/OR MODIFICATION OF THE FABRICATED ELECTRONIC SYSTEM
1806
OPERATE FABRICATED ELECTRONIC SYSTEM IN A TARGET ENVIRONMENT AND AT SPEED
1808
PERFORM HDL-BASED HARDWARE DEBUGGING ON THE OPERATING FABRICATED ELECTRONIC SYSTEM
1810
END

FIG. 18

FIG. 19-1


1900
BREAK-POINT DATABASE
602
BREAK-POINT MANAGER
1906
ACTIVE BREAK-POINT DATABASE
1904
ACTIVE SIGNAL VALUE DATABASE
1908
SIGNAL VALUE MANAGER
1910
SIGNAL VALUE DATABASE
604
HDL DESC.
304
SELECTION USER INTERFACE
1902
DIC DATABASE
736
DIC CONFIGURATION MANAGER
1916
DIC CONFIG. DATABASE
1918
DIC STATUS DATABASE
1928
R2P TRANSLATOR
1920
RAW CONFIG. FILE
1922
RAW STATUS FILE
1926
HOST COMM. CNTLR.
1924
RUN-TIME CONTROLLER
1912
RUN-TIME USER INTERFACE
1914
A
B
C

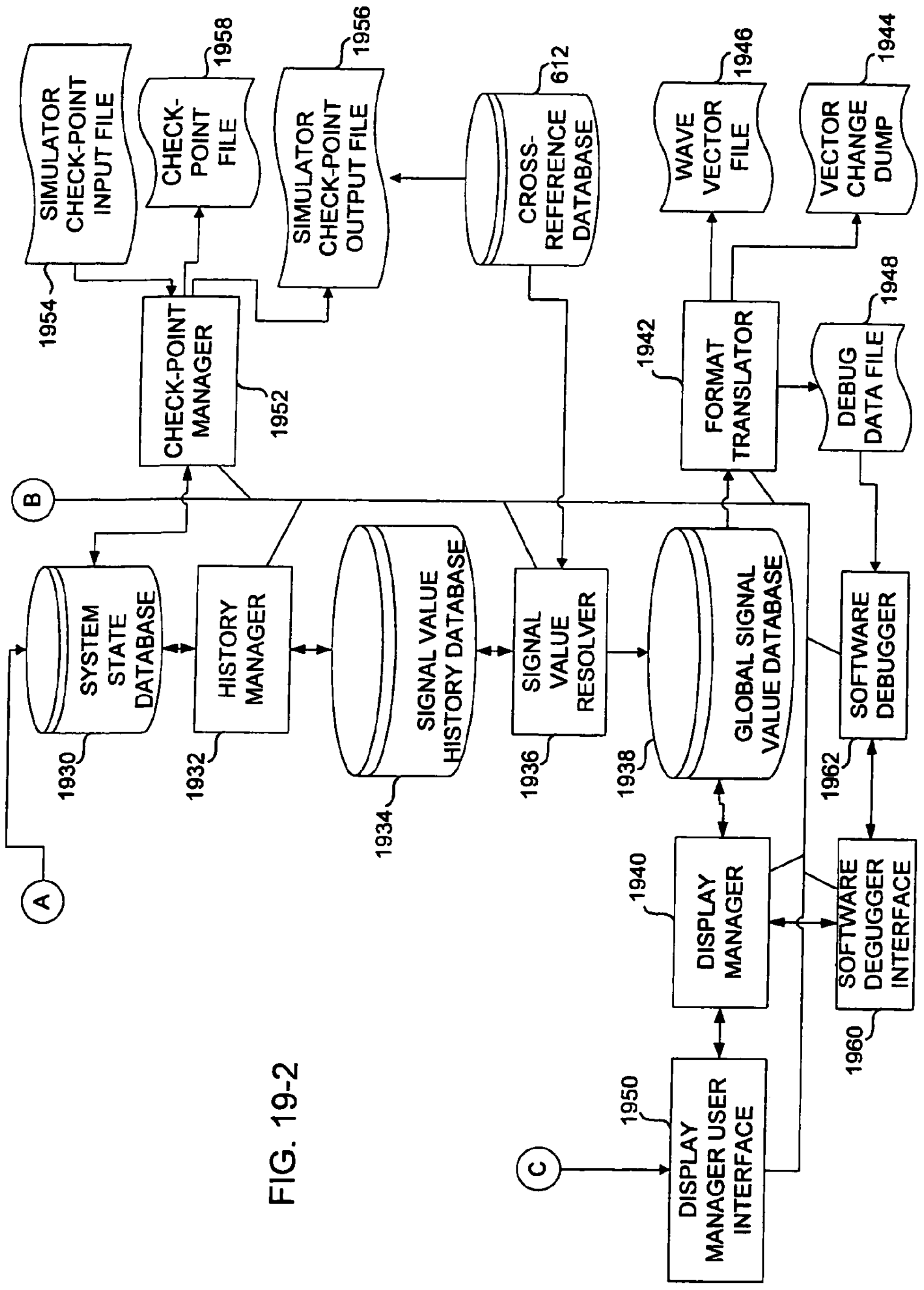

SIMULATOR CHECK-POINT INPUT FILE
1954
CHECK-POINT FILE
1958
SIMULATOR CHECK-POINT OUTPUT FILE
1956
CROSS-REFERENCE DATABASE
612
WAVE VECTOR FILE
1946
VECTOR CHANGE DUMP
1944
CHECK-POINT MANAGER
1952
FORMAT TRANSLATOR
1942
DEBUG DATA FILE
1948
B
SYSTEM STATE DATABASE
1930
HISTORY MANAGER
1932
SIGNAL VALUE HISTORY DATABASE
1934
SIGNAL VALUE RESOLVER
1936
GLOBAL SIGNAL VALUE DATABASE
1938
SOFTWARE DEBUGGER
1962
A
DISPLAY MANAGER
1940
SOFTWARE DEGUGGER INTERFACE
1960
C
DISPLAY MANAGER USER INTERFACE
1950

FIG. 19-2

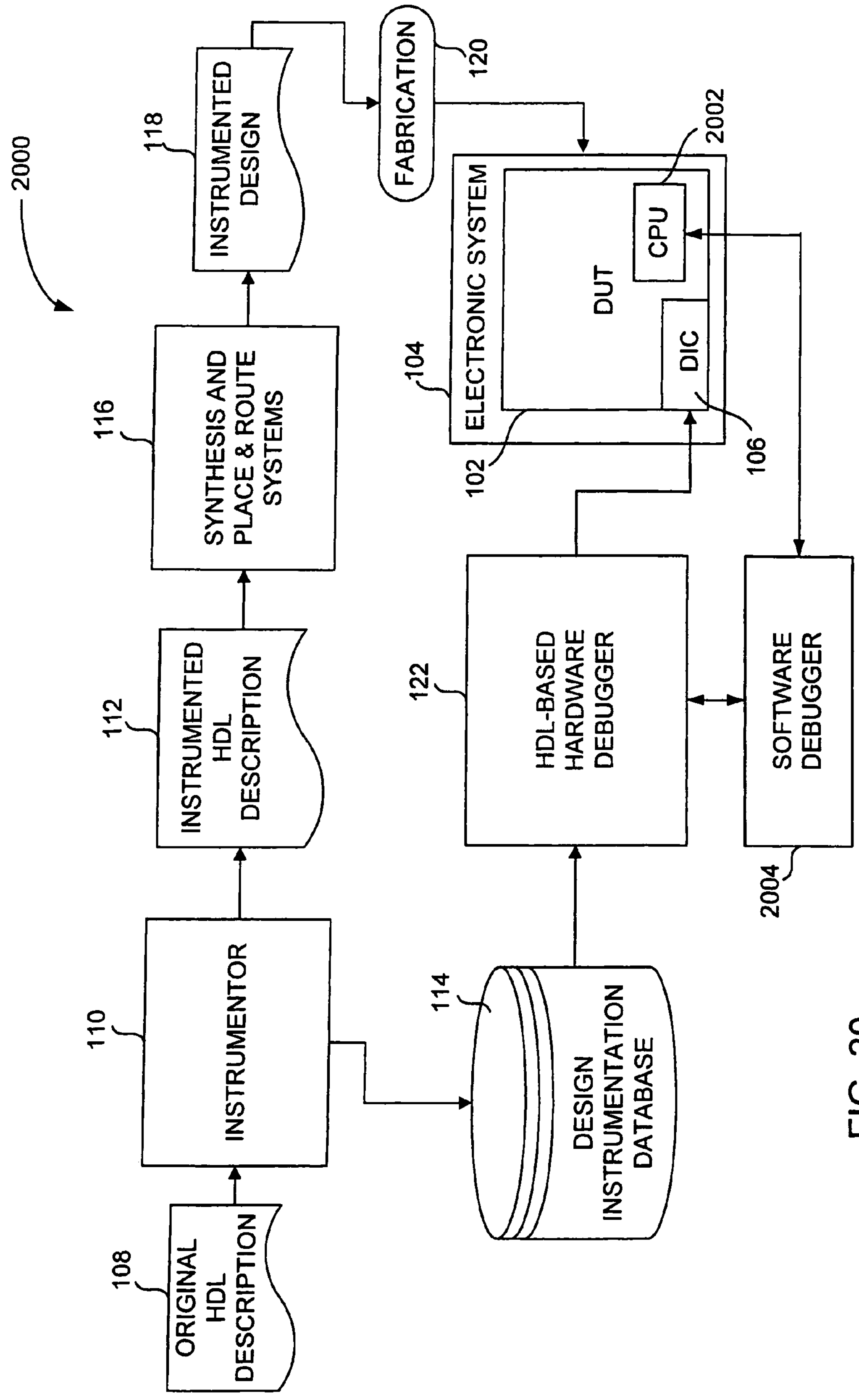
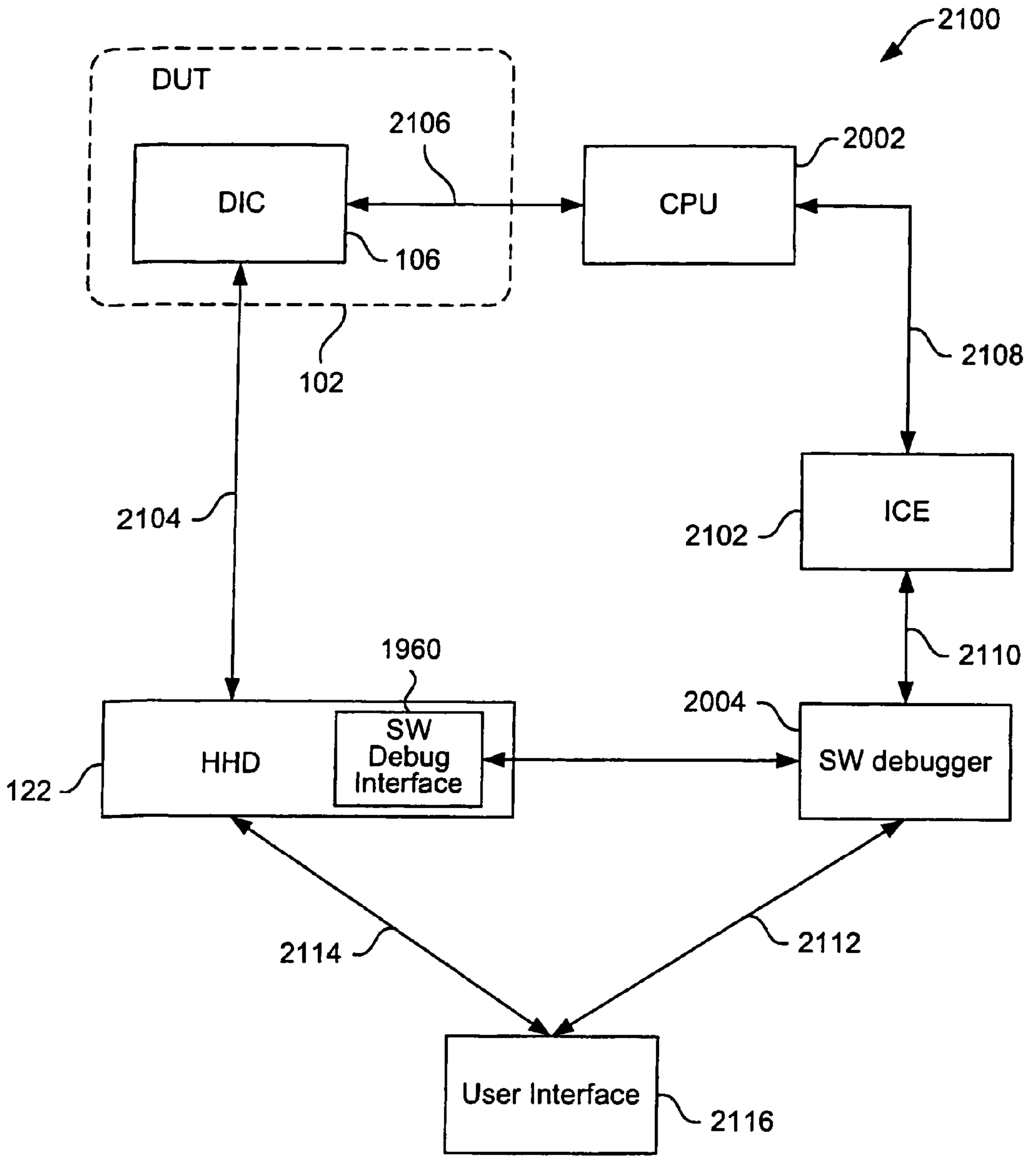

2000
108
ORIGINAL HDL DESCRIPTION
110
INSTRUMENTOR
112
INSTRUMENTED HDL DESCRIPTION
116
SYNTHESIS AND PLACE & ROUTE SYSTEMS
118
INSTRUMENTED DESIGN
FABRICATION
120
114
DESIGN INSTRUMENTATION DATABASE
122
HDL-BASED HARDWARE DEBUGGER
104
ELECTRONIC SYSTEM
DUT
CPU
2002
DIC
102
106
SOFTWARE DEBUGGER
2004

FIG. 20



2100
DUT
DIC
106
102
2106
CPU
2002
2108
ICE
2102
2110
2104
1960
HHD
SW Debug Interface
122
SW debugger
2004
2114
2112
User Interface
2116

FIG. 21

HARDWARE DEBUGGING IN A HARDWARE DESCRIPTION LANGUAGE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation application of U.S. patent application Ser. No. 10/406,732, filed on Apr. 2, 2003, now U.S. Pat. No. 6,904,577 which is a continuation of U.S. patent application Ser. No. 09/724,702 filed on Nov. 28, 2000, now issued as U.S. Pat. No. 6,581,191. This application also claims the benefit of: (i) U.S. Provisional Patent Application No. 60/168,266, filed Nov. 30, 1999, and entitled "INTERACTIVE DEBUGGING OF HDL SOURCE CODE", and which is hereby incorporated by reference herein; and (ii) U.S. Provisional Patent Application No. 60/230,068, filed Aug. 31, 2000, and entitled "HDL-BASED HARDWARE DEBUGGING", and which is hereby incorporated by reference herein.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to electronic systems and, more particularly, to debugging of electronic systems.

2. Description of the Related Art

Electronic systems are designed by designers to operate in specific ways. Electronic systems are systems that contain digital and/or analog electronic components connected together to perform specific operations or functions. Besides the electronic components, electronic systems may also include software. Once designed, the electronic systems may need to be debugged. Debugging electronic systems is a process which involves detection, diagnosis, and correction of functional failures. In the detection step, the designer of the electronic system observes a functional failure. When the designer is able to gather enough information about the incorrect behavior of the electronic system, the designer of the electronic system can draw the necessary conclusions to diagnose the functional failure. For correction of the functional failure, a fix is applied and subsequently tested. When the design is provided in a Hardware Description Language (HDL), such a fix may be a textual change to the HDL description of the electronic system.

In general, debugging has conventionally been performed by various different approaches. In particular, debugging has been performed by computer software debugging, hardware description language functional verification, hardware logic level analysis, or hardware behavioral source level emulation. These different approaches are discussed below.

Computer software debugging is conventionally performed using a computer software debugger. A computer software debugger is a software tool that allows a software developer to control the execution of a running computer software program by setting break-points, sequentially single-stepping through the execution of the computer software program, and looking at the program's state by examining and displaying variables and expressions. One example of such a software debugging tool is the GNU Debugger (GDB), which can be obtained from Red Hat, Inc. in Sunnyvale, Calif.

Software debuggers usually offer interactive debugging of software programs which are sequentially executed on computers. However, some software debuggers also support limited concurrency such as threaded program execution. Some software debuggers support debugging programs written at different levels of abstraction from high-level computer languages such as C++ down to assembler code and/or machine code. To assist with debugging of programs written in high-level computer languages, the software debugging system can add extra debug information (e.g., symbolic names and references to source code) to the compiled code during compilation of the computer software program. In combination with in-circuit emulators, software debuggers may provide a limited capability to analyze the underlying Central Processing Unit (CPU) of the computer executing the computer software program. A major disadvantage of software debuggers is, however, that they cannot be used for efficiently debugging general hardware of electronic systems.

Hardware description language functional verification is used to verify that the parts of an electronic system which are described using HDL match their functional specification. Such functional verification can be achieved through functional simulation or formal verification.

Functional simulation is performed by a functional simulator. A functional simulator is a software program that runs on a host computer and simulates the operation of an electronic system using its HDL description. Examples of functional simulators include VCS and VSS from Synopsys, Inc. in Mountain View, Calif., and ModelSim from Mentor Graphics Corp. in Wilsonville, Oreg. To increase simulation performance some functional simulators additionally make use of special purpose hardware which acts as a co-processor and accelerates the simulation. An example of a hardware-accelerated functional simulator is the Hammer system from Tharas Systems, Inc. in Santa Clara, Calif. Unfortunately, one major disadvantage of functional simulation is the need for simulation models. In order to be able to simulate, there must exist a simulation model with the proper functional behavior for each component of the HDL design for the electronic system. For some components such simulation models may not be readily available and must be generated. Additionally, the HDL design must be stimulated by a testbench. Since the ideal testbench must correctly and exhaustively match the behavior of the target environment, creation of a testbench can be very difficult and time consuming. On the other hand, a testbench that is too simple will not provide the necessary coverage to find all the design errors. Although functional simulation is useful, using functional simulation to debug design errors is too burdensome. Not only are the testbenches difficult to create, but also the more complex the HDL design is, the lower the performance of functional simulation. For state-of-the-art HDL designs simulation is now a million times slower than the fabricated hardware. Hardware-acceleration can typically speedup functional simulation by a factor on the order of one-hundred. Accordingly, its low performance makes it impractical to use functional simulation either to debug real-time applications or to concurrently debug hardware and software of complex electronic systems.

Formal verification is performed by a formal verification tool. Formal verification can help with the problem of incomplete coverage in functional simulation due to testbench limitations. One approach checks the HDL description for properties. Such properties may be explicitly provided by the designer of the electronic system or implicitly extracted from the HDL description by the formal verification tool. An example of such a formal verification tool is Solidify from Averant, Inc. in Sunnyvale, Calif. One disadvantage of formal verification is that it is impractical to use to re-produce functional failures observed in a running electronic system.

Both techniques, functional simulation and formal verification, have the major disadvantage that they do not operate on fabricated hardware. Instead, both techniques operate on a model of the electronic system and a model of the environment in which the electronic system runs, i.e., a testbench. Thus, their use is limited to debugging design errors. As such, neither technique is applicable for debugging manufacturing faults, environment errors, timing errors and/or tool errors. Also, inadequacies in the testbench have the potential to hide or introduce design errors in the HDL design during functional simulation which can later, when the HDL design is fabricated, show up as functional failures of the running electronic system.

Hardware logic level analysis is a technique that works at the logic level of a fabricated electronic system. The logic level of abstraction is also referred to as gate-level. Since electronic systems have been designed at the logic level for many years (for example using schematic entry of logic gates and flip-flops), there exists a wide variety of different techniques for debugging at logic level, including: digital logic is analyzers, in-circuit emulators, Design-For-Test (DFT) techniques, and hardware emulation, each of these different techniques are discussed below.

Digital logic analyzers operate to probe a limited number of digital signals and record their logic values. Probing is accomplished by physically connecting probes of the digital logic analyzer to exposed pins and/or circuitry on the fabricated design. Recording is controlled by trigger conditions, which are conditional expressions built upon the values of the recorded signals provided by the probes. The values for the recorded signals are stored in dedicated memory inside the digital logic analyzer so as to be available for subsequent display. Digital logic analyzers can be external devices or blocks embedded inside the digital circuits of an electronic system. An example of an external digital logic analyzer is the Agilent 16715A from Agilent Technologies, Inc. in Palo Alto, Calif. Examples of embedded logic analyzers are SignalTap from Altera Corporation in San Jose, Calif., or ChipScope from xilinx, Inc. in San Jose, Calif. Another example of an embedded logic analyzer was presented at the 1999 IEEE International Test Conference by Bulent Dervisoglu in "Design for Testability: It is time to deliver it for Time-to-Market". Since embedded logic analyzers are added to the circuitry of the design, they can probe internal signals. Thus, embedded digital logic analyzers overcome the limited access to internal signals problem of external logic analyzers because access to the internal signals is not restricted by the pins of the fabricated circuits.

An in-circuit emulator is a specialized piece of hardware that connects to a CPU for debugging the CPU and the software that runs on the CPU. An example of an in-circuit emulator is visionICE from Windriver in Alameda, Calif. However, since in-circuit emulators only work for the specific target CPU for which they were built, in-circuit emulators are inappropriate for debugging general digital circuits.

DFT techniques, such as boundary scan and built-in self test, provide access to the internal registers of a running fabricated digital circuit. An example of such technique is described in the IEEE 1149.1 JTAG standard available from the Institute of Electrical and Electronic Engineers in Piscataway, N.J. DFT techniques are also described in "Digital Logic Testing and Simulation" by Alexander Miczo, published by Wiley, John and Sons Inc., 1985. DFT techniques were originally developed for and applied to testing of manufacturing faults and have the major disadvantage that they do not relate back to the HDL description.

Hardware emulation systems map a synthesized HDL design onto special emulation hardware. Such emulation hardware comprises many re-programmable FPGA devices and/or special purpose processors. The emulation hardware then executes a model of the HDL design. Thus hardware emulation has the same disadvantage as functional simulation, namely, that it works on a model of the electronic system and not on the fabricated hardware. As a result, hardware emulation systems are limited to design error debugging, and cannot be used for diagnosing manufacturing faults, tool errors, timing errors, etc. An example of such a hardware emulation system is System Realizer from Quicktum Systems, in San Jose, Calif. Specially built prototyping systems comprising FPGAs/PLDs can also be seen as hardware emulation systems. Since hardware emulation is usually much faster than functional simulation, hardware emulation systems may enable use of the software that is supposed to run on the HDL design to be used as a testbench. Even so, hardware emulation typically runs at speeds below one MegaHertz (MHz) while the HDL design is supposed to run at many hundred MegaHertz. In some cases the emulator speed may allow the user to connect the HDL design to the target environment which makes the design of testbenches unnecessary. Even so, with the high speeds of state-of-the-art HDL designs, hardware emulation is not capable of debugging the majority of real-time applications. Another disadvantage is that the special synthesis, mapping, and multi-chip partitioning steps required to bring an HDL design into a hardware emulation system are very complicated and time consuming.

A major drawback of all logic level debugging techniques is that they work at the logic level of abstraction. Since the HDL-based design methodology of electronic systems is much more efficient for todays complex designs, HDL designs have largely replaced logic level designs. Application of logic level debugging techniques to HDL design methodology is highly inefficient. Since logic level debugging does not-relate back to the HDL description, it normally would not provide the designer of the electronic system with sufficient information to correctly diagnose a functional failure.

Hardware behavioral source level emulation provides hardware emulation of source level designs. One technique for debugging HDL designs described at the behavioral level HDL using hardware emulation is described in "Interaktives Debugging algorithmischer Hardware-Verhaltensbeschreibungen mit Emulation" by Gemot H. Koch, Shaker Verlag, Germany, 1998. Some of which is also described in Koch et al., "Breakpoints and Breakpoint Detection in Source Level Emulation," ACM Transactions on Design Automation of Electronic Systems, Vol. 3, No. 2, 1998. The therein described technique is referred to as Source Level Emulation (SLE) and offers an approach for emulating HDL designs, however only if such designs are described in behavioral VHDL. During behavioral synthesis a behavioral HDL design is enhanced for debugging by generating and adding additional circuitry for break-point detection. The behavioral synthesis tool writes out synthesized VHDL which contains a register transfer level description of the enhanced HDL design. The register transfer level description is then synthesized, mapped, and multi-chip partitioned into the emulation hardware. During hardware emulation with a hardware model of the HDL design, the user is able to examine particular variables in the behavioral HDL description.

Control is provided via break-points which are detected using the additional circuitry inside the running hardware model. Break-points in SLE have a very specific meaning. In particular, such break-points are closely tied to behavioral operations in the data-flow of the behavioral HDL description, and are associated with particular states of a controller which is generated by the behavioral synthesis. Additionally, break-points can be made conditioned upon particular values of data-path registers. When a break-point is detected, the execution of the hardware model is stopped. This is done by halting some or all of the system clocks and prevents the registers from changing their current values. Once halted, internal registers can be read. These registers form a scan-chain such that their values can be read by an emulation debugging tool.

Examination of variables in the behavioral HDL description is done in two ways. For variables which are mapped by the behavioral synthesis into registers in the hardware model, their values can be read and related back to HDL identifiers. This is done using map files which keep track of the transformations in behavioral synthesis, register transfer level synthesis, mapping, and multi-chip partitioning. For variables which have not been mapped to registers in the hardware model, their values are computed using a functional model of the behavioral HDL design. This functional model is created during behavioral synthesis and requires the existence of functional models of its components. The values, either read or computed, are then displayed in the behavioral HDL description. Optionally, by overwriting some or all of the registers of the hardware model while the hardware model is halted, the behavior of the HDL design can be modified once the execution of the hardware model is resumed.

Although source level emulation provides a debugging method which works at the level of the HDL description (in this case behavioral VHDL), it has various drawbacks which limits its use in practice. Several of the drawbacks are as follows. First, enhancements for source level emulation must be done inside a behavioral synthesis tool, since it needs special information about the behavioral HDL design which is only available during the behavioral synthesis process. Second, source level emulation does not allow the designer to perform customization. For example, a designer is not able to select trade-offs between hardware overhead and debugging support. Third, source level emulation cannot handle HDL descriptions on levels of abstraction other than the one provided by behavioral VHDL. Explicitly, source level emulation is not applicable for the most commonly used levels of abstraction of RTL HDL and gate-level HDL. Fourth, source level emulation supports neither hierarchy nor re-use of pre-designed blocks. Fifth, there are various limitations and difficulties in relating registers back to behavioral HDL source code. Sixth, in order to examine the state of the hardware model, it is required that some or all of the system clocks be halted and the hardware stopped, which makes source level emulation inapplicable for debugging the majority of today's electronic systems which are not to be stopped.

Thus, there is a need for efficient and effective approaches for debugging HDL-based electronic system designs.

SUMMARY OF THE INVENTION

Broadly speaking, the invention relates to techniques and systems for analysis, diagnosis and debugging fabricated hardware designs at a Hardware Description Language (HDL) level. Although the hardware designs (which were designed in HDL) have been fabricated in integrated circuit products with limited input/output pins, the invention

enables the hardware designs within the integrated circuit products to be comprehensively analyzed, diagnosed, and debugged at the HDL level at speed. The ability to debug hardware designs at the HDL level facilitates correction or adjustment of the HDL description of the hardware designs.

The invention can be implemented in numerous ways including, a method, system, device, and computer readable medium. Several embodiments of the invention are discussed below.

As a hardware debugging system for debugging a fabricated integrated circuit containing an electronic circuit design, one embodiment of the invention includes at least: an instrumentor configured to receive a high level HDL description of the electronic circuit design, to determine aspects of the electronic circuit design to be examined or modified during debugging, to determine additional circuitry to be incorporated into the electronic circuit design to facilitate debugging, and to produce a modified high level HDL description of the electronic circuit design by incorporating an HDL description of the additional circuitry into the high level HDL description of the electronic circuit-design; a design instrumentation database configured to store information about the additional circuitry including relationships between signals of the electronic circuit design and portions of the modified high level HDL description or the high level HDL description; and a HDL-based hardware debugger configured to debug the fabricated integrated circuit fabricated in accordance with the modified high level HDL description by interacting with the electronic circuit design using the additional circuitry and by operating to present debug information with respect to the modified high level HDL description or the high level HDL description.

As a hardware debugging system for debugging an electronic system containing an electronic circuit design, the electronic circuit design being described by a high level HDL description, one embodiment of the invention includes at least: an instrurentor configured to receive the high level HDL description of the electronic circuit design or a description derived therefrom, to determine aspects of the electronic circuit design to be examined or modified during debugging, to determine additional circuitry to be incorporated into the electronic circuit design to facilitate debugging, and to incorporate the additional circuitry into the electronic circuit design; a design instrumentation database configured to store information about the additional circuitry including relationships between signals of the electronic circuit design and portions of the high level HDL description; and a HDL-based hardware debugger configured to debug the electronic system by interacting with the electronic circuit design using the additional circuitry and by operating to present debug information with respect to the high level HDL description.

As a hardware debugging system for debugging an electronic system containing an electronic circuit design, the electronic circuit design being described by a high level HDL description, another embodiment of the invention includes at least: instrumentation means for receiving the high level HDL description of the electronic circuit design or a description derived therefrom, determining additional circuitry to be incorporated into the electronic circuit design to facilitate debugging, and incorporating the additional circuitry into the electronic circuit design; a design instrumentation database configured to store information about the additional circuitry including relationships between signals of the electronic circuit design and portions of the high level HDL description; and a HDL-based hardware debugger configured to debug the electronic system by interacting with the electronic circuit design using the additional circuitry and by operating to present debug information with respect to the high level HDL description.

Other aspects and advantages of the invention will become apparent from the following detailed description taken in conjunction with the accompanying drawings which illustrate, by way of example, the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

FIG. **1**A is a block diagram of a hardware debugging system according to one embodiment of the invention;

FIG. **1**B is a block diagram of a hardware debugging system according to another embodiment of the invention;

FIG. **2** is a flow diagram of basic instrumentation processing according to one embodiment of the invention;

FIG. **3** is a block diagram of an instrumentation system according to one embodiment of the invention;

FIGS. **4**A and **4**B are flow diagrams of detailed design instrumentation processing according to one embodiment of the invention;

FIG. **5**A is a flow diagram of selection processing according to one embodiment of the invention;

FIG. **5**B is a flow diagram of break-point processing according to one embodiment of the invention;

FIG. **5**C is a flow diagram of explicit trigger condition selection processing according to one embodiment of the invention;

FIG. **5**D is a flow diagram of sampling signal selection processing according to one embodiment of the invention;

FIG. **6** is a diagram of a design instrumentation database according to one embodiment of the invention;

FIG. **7**A is a block diagram of an instrumentation system according to one embodiment of the invention;

FIG. **7**B is a diagram of a hard block resolution system according to one embodiment of the invention;

FIG. **8** is a block diagram of a representative Design Instrumentation Circuit (DIC) according to one embodiment of the invention;

FIG. **9** describes a representative generic configurable circuitry which can implement design sampling and design patching according to one embodiment of the invention;

FIG. **10** illustrates a representative generic configurable trigger detection circuit according to one embodiment of the invention;

FIG. **11** illustrates a representative state based Finite State Machine design control circuit according to one embodiment of the invention;

FIG. **12** illustrates a representative transition based Finite State Machine design control circuit according to one embodiment of the invention;

FIG. **13** illustrates a representative data-path register design control circuit according to one embodiment of the invention;

FIG. **14** illustrates a representative part of the design control circuit according to one embodiment of the invention;

FIG. **15** is a block diagram of a portion of an instrumentation system which includes a cross-reference analysis module and a cross-reference database according to one embodiment of the invention;

FIG. **16** is a block diagram of a portion of an instrumentation system which includes a DFT analysis module according to one embodiment of the invention;

FIG. **17** is a data flow diagram illustrating DIC creation processing according to one embodiment of the invention;

FIG. **18** is a flow diagram of HDL-based hardware debugging processing according to one embodiment of the invention;

FIG. **19** is a data flow diagram of a debugging process performed by a HDL-based hardware debugger according to one embodiment of the invention;

FIG. **20** is a block diagram of a hardware/software co-debugging system according to one embodiment of the invention; and

FIG. **21** is a block diagram of a hardware/software co-debugging system according to one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates to techniques and systems for analysis, diagnosis and debugging fabricated hardware designs at a Hardware Description Language (HDL) level. Although the hardware designs (which were designed in HDL) have been fabricated in integrated circuit products with limited input/output pins, the invention enables the hardware designs within the integrated circuit products to be comprehensively analyzed, diagnosed, and debugged at the HDL level at speed. The ability to debug hardware designs at the HDL level facilitates correction or adjustment to the HDL of the hardware designs.

The following discussions will be made clearer by a brief review of the relevant terminology as it is typically (but not exclusively) used. Accordingly, to assist readers in understanding the terminology used herein, the following definitions are provided.

"Software" is defined as but not limited to programming language content written using a programming language. Examples of programming languages include C, C++, Basic, assembly, and Java.

"HDL" is a Hardware Description Language. A hardware description language is defined as any programming language that can describe the hardware portion of an electronic system. Examples of HDLs include VHDL which is described in the IEEE Standard VHDL Language Reference Manual available from the Institute of Electrical and Electronic Engineers in Piscataway, N.J., which is hereby incorporated by reference; Verilog HDL which is described in Hardware Modeling with Verilog HDL by Eliezer Stemnheim, Rajvir Singh, and Yatin Trivedi, published by Automata Publishing Company, Palo Alto, Calif., 1990, which is hereby incorporated by reference; and SystemC which stems from the Open SystemC Initiative (OSCI) originally started by Synopsys, Inc. of Mountain View, Calif. General purpose programming languages such as C++, C, and assembly languages may also be used as a HDL.

A "high level HDL description" is a HDL description in which at least a portion of the description is at register transfer level (RTL) or higher. For VHDL this synthesizable, register transfer level subset is described in the IEEE 1076.6-1999 Standard for VHDL Register Transfer Level (RTL) Synthesis, available from the Institute of Electrical and Electronic Engineers in Piscataway, N. J., which is hereby incorporated by reference. The synthesizable register transfer level subset of the Verilog HDL is described in

"Verilog HDL: A Guide to Digital Design and Synthesis" by Samir Palnitkar, SunSoft Press, 1996.

A "RAM" is a Random Access Memory—defined as an electronic component capable of storing data.

"ASIC" is an Application Specific Integrated Circuit. An ASIC device is an electronic component of a system. ASICs are custom devices created for a specific 5 purpose within the electronic system. ASIC devices are easier and faster to create with respect to a full custom semiconductor device. An ASIC may be described using HDL and implemented using synthesis.

An "FPGA" is a Field Programmable Gate Array. FPGAs are electronic components that have a configurable function. These devices are able to change their 10 functionality via an information stream transferred to the device. These electronic components are available from a number of different suppliers in a wide range of sizes and speeds. One example of these devices are the Virtex FPGA devices from Xilinx, Inc. located in San Jose, Calif. An FPGA design may be described using HDL and implemented using synthesis.

A "Central Processing Unit" or "CPU" is circuitry controlling the interpretation and execution of software programmed instructions, performs arithmetic and logical operations on data, and controls input/output functions. For the following descriptions the term CPU will be used to also denote other processing elements such as microprocessors, digital signal processors, microcontrollers, etc.

A "register" is an element in digital circuitry which can store one or more bits. Examples for registers are the various types of flip-flops and latches.

A "PLD" is an Programmable Logic Device. PLDs are electronic components that have a configurable function. These devices are able to change their functionality via an information stream transferred to the device. These electronic components are available from a number of different suppliers in a wide range of sizes and speeds. One example of these devices are the Apex PLD devices from Altera Corporation in San Jose, Calif. A PLD design may be described using HDL and implemented using synthesis.

"Electronic Components" are defined as but not limited to, transistors, logic gates, integrated circuits, semi-custom integrated circuits, full custom integrated circuits, application specific integrated circuits (ASICs), gate arrays, programmable logic devices (PLDs), field prograrrmable gate arrays (FPGAs), CPUs, Random Access Memory (RAM), mixed signal integrated circuits, systems on a chip (SOC), and systems on a printed circuit board.

An "Electronic System" is defined as a system that contains one or more digital and/or analog Electronic Components connected together to perform specific operations or functions. An Electronic System can be implemented entirely of hardware (Electronic Components) or consist of a mix of hardware and software (programming language content).

"Mixed-signal designs" are defined as Electronic System designs which incorporate both digital and analog signals.

The "HDL Design" is referred to as the portion of the electronic system which is described in HDL and implemented in hardware. It is also referred to as the "Design under Test" (DUT).

An "SOC" is a System On Chip. A SOC is defined as a device large enough to contain an entire electronic system implementation. SOC devices can integrate a number of electronic devices into one device.

An "HDL description" is the textual description of an HDL Design. "HDL source code" is referring to the text files which contain the HDL description.

An "HDL identifier" is an object in an HDL description which represents a signal, a set of signals, a storage element, or a set of storage elements and which can take a value from a set of possible values. Each HDL identifier is associated with a particular scope defined by the syntax of the umderlying hardware description language.

A "Technology Mapping Process" is defined as the process of transforming a particular representation of an electronic design into a design consisting entirely of primitive electronic elements from a design library for a target technology. The representation of said electronic design from which the Technology Mapping Process begins is dependent on the particular Technology Mapping Process being employed. However, said representation usually consists of simple boolean elements. For example, said representation may consist entirely of an interconnected set of 2-input/1-output logic elements with each said element performing the NAND function. An example of a tool that performs the Technology Mapping Process is Design Compiler from Synopsys in Mountain View, Calif.

"Synthesis" is defined as the process of creating an electronic implementation from the functional description of a system. An example of a tool that performs this operation is Design Compiler from Synopsys in Mountain View, Calif. which reads electronic system descriptions written in a synthesizable subset of VHDL and Verilog and produces a technology mapped design as an output.

"Behavioral HDL" is an HDL description at an algorithmic level of abstraction in which neither the timing behavior nor the structure of the HDL Design is explicitly described.

"Behavioral synthesis" transforms a behavioral HDL description into a register transfer level (RTL) description where the timing behavior and the structure of the HDL Design is fixed. This RTL description is then processed in synthesis and technology mapping. An example of a tool that performs behavioral synthesis is Behavioral Compiler from Synopsys, Inc. of Mountain View, Calif.

A "System Clock" is defined as a main timekeeping signal in a design. All operations that are relative to the System Clock will proceed when the System Clock becomes active.

"Constraints" are defined as limits placed on parameters for the implementation of an electronic system. Parameters of an electronic system can include but are not limited to register to register propagation delay, system clock frequency, primitive element count, and power consumption. These constraints can be used by synthesis tools to guide the implementation of the electronic design.

"Fabrication" is the process of transforming a synthesized and technology mapped design into one or more devices of the target technology. For example, the fabrication of ASICs involves manufacturing and the fabrication of FPGAs and PLDs involves device configuration.

"DFT" is Design-for-test. DFT is defined as a process in which an electronic system designer will include structures in the electronic system that facilitates manufacturing testing.

"Design Rule Check" (DRC) are checks performed before integrated circuit manufacturing to ensure that in the placed and routed technology mapped design none of the rules of the target technology process is violated. Examples for such DRC are checks for shorts, spacing violations, or other design-rule problems between logic cells. An example for a tool that does DRC is Dracula from Cadence Design Systems, Inc. in San Jose, Calif.

A "Functional Specification" is defined as the documentation that describes the necessary features and operations of a system.

A "functional failure" is a behavior of a design which does not meet the functional specification which was used in the creation of the design. Every step in the design process can potentially cause a functional failure. Functional failures can be classified depending on which step of the design process caused the functional failure.

A "fault" is a specific type of functional failure. This type of failure is due to one or more manufacturing defects causing a functional failure in the fabricated design.

A "design error" is a specific type of functional failure where the HDL description's behavior did not match the functional specification.

A "tool error" is a specific type of functional failure which was introduced by design tools because the HDL description was not properly processed such that the functional specification is not met by the implementation.

An "environment error" is a specific type of functional failure caused by a particular combination of environmental parameters such as temperature, humidity, pressure, etc.

A "Functional Simulator" is a tool that mimics the functional behavior of a model of an electronic system which is described using HDL.

A "Testbench" is defined as an electronic system description that presents stimulus to and/or gathers information from the target electronic system design to be verified. In some cases the testbench ignores the response from the target electronic system design. A testbench is used to mimic the behavior of the target environment in which the electronic system being developed will operate. A testbench may comprise both hardware and software.

A "Target Environment" is the system the electronic system is specified to interact with and/or to run in. A target environment may comprise both hardware and software.

The "Target Speed" of an electronic system is the speed and/or the speed range the electronic system is specified to run at. Examples for measures for the target speed and the speed range are clock frequency, response time, time to propagate, and cycle time.

"Debugging" is the process of comparing the behavior of an implementation of the electronic system to the electronic system functional specification. The purpose of debugging is to find causes and remedies for functional failures.

"Co-Debugging" or "hardware/software co-debugging" is defined as the process of debugging the software and hardware of an electronic system concurrently.

A "FSM" is Finite State Machine—defined as an electronic system control structure. The design and implementation of FSM is described in great detail in Synthesis and Optimization of Digital Circuits, by Giovanni DeMicheli, McGraw Hill, 1994.

A "HDL Building Block" is a functional unit of an HDL Design from which the HDL Design is constructed. A HDL Building Block (BB) performs calculations on the signals to which it is connected and communicates with other BBs in the design. The communication is through connecting internal signals of a BB to communication ports of the BB and/or connecting internal signals of the BB to communication ports of other BBs in the HDL Design. Examples of BBs are Entities in the VHDL language and Modules in the Verilog language.

A "Hard Block" is an electronic system which has a pre-defined functionality and which can be incorporated into another electronic system. Commonly, the form of the Hard Block is such that the functionality of the Hard Block can not be altered. An example of a hard block is an HDL Design which implements a industry standard bus controller.

A "Design State" is defined as the logical values taken by the storage elements of the design at a particular time, combined with the logical values taken by the inputs of the design taken at the same particular time.

The "System State" or "State of the System" is a synonym for "Design State." "Real-time" means a task, process or response occurs substantially immediately. The term is used to describe a number of different computer features. For example, real-time operating systems are systems that respond to input immediately. Real-time is also used for describing tasks in which the computer must react to a steady flow of new information without interruption. Real-time can also refer to events simulated by a computer at the same speed that they would occur in real life.

Embodiments of this aspect of the invention are discussed below with reference to FIGS. **1**–**21**. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes as the invention extends beyond these limited embodiments.

FIG. **1**A is a block diagram of a hardware debugging system **100** according to one embodiment of the invention. The hardware debugging system **100** operates to debug a hardware product referred to herein as a Device Under Test (DUT) **102**. The DUT **102** is typically part of a larger hardware product referred to as an electronic system **104**. The DUT **102** can pertain to a single integrated circuit chip, multiple integrated circuit chips, a system on a chip, or a system on a printed circuit board.

According to the invention, the DUT **102** includes Design Instrumentation Circuitry (DIC) **106**. The DIC **106** is provided within the DUT **102** in order to facilitate debugging of the DUT **102**. The DIC **106** can be provided within the DUT **106** in either a centralized or distributed manner.

The hardware debugging system **100** operates to determine the DIC **106** that is provided within the DUT **102**. In this regard, an original HDL description **108** of the electronic system **104** is received at an instrumentor **110**. The instrumentor **110** modifies or alters the original HDL description **108** to produce an instrumented HDL description **112**. The instrumented HDL description **112** represents not only the electronic system **104** with the DUT **102** provided therein, but also the DIC **106** that is provided within the DUT **102**. The instrumentor **110** also stores DIC information to a design instrumentation database **114**. By storing the DIC information in the design instrumentation database **114**, the hardware-based debugging of the DUT **102** is facilitated.

The hardware debugging system **100** also includes synthesis and place & route systems **116**. The synthesis and place & route systems **116** receives the instrumented HDL description **112** and performs conventional synthesis as well as place & route operations in order to produce an instrumented design **118**. Although not shown in FIG. **1**A, other additional tools can be utilized to produce or enhance the instrumented design **118**. Examples of additional tools include a Design-For-Test (DFT) tool or a Design Rule Check (DRC) tool. The instrumented design **118** represents a description (e.g., design files) of the electronic system **104** that would be thereafter fabricated. Hence, once the instrumented design **118** is available, fabrication **120** can be performed. The fabrication **120** produces the electronic system **104** having the DUT **102** with the DIC **106** provided therein. Fabrication is the process of transforming a synthesized and technology mapped design (e.g., the instrumented design **118**) into one or more devices of the target technology. For example, if the target technology is Application Specific Integrated Circuits (ASICs) then the fabrication involves manufacturing, and if the target technology is Field Programmable Gate Arrays (FPGAs) or Programmable Logic Devices (PLDs) the fabrication involves device configuration.

At this point, the electronic system **104** is a hardware product that has been produced. This hardware product can then be debugged using a HDL-based hardware debugger **122**. More particularly, the HDL-based hardware debugger **122** couples to the DIC **106** so that it is able to communicate with the DIC **106** when debugging the DUT **102**. The HDL-based hardware debugger **122** also couples to the design instrumentation database **114** so that access to the DIC information is available. As a result, the HDL-based hardware debugger **122** enables a user to debug the DUT **102** and/or hardware and/or software interacting with the DUT **102** in close relation to the original HDL description **108**. Further, in one embodiment, debugging can be performed while the electronic system **104** and the DUT **102** operate in the target environment, at target speed.

FIG. **1**B is a block diagram of a hardware debugging system **150** according to another embodiment of the invention. The hardware debugging system **150** is similar to the hardware debugging system **100** and includes many of the same components. Hence, the hardware debugging system **150** enables a user of the HDL-based hardware debugger **122** to debug the DUT **102** of the electronic system **104** and/or hardware and/or software interacting with the DUT **102**, as noted above. However, the hardware debugging system **150** includes a synthesis and place & route system **152** that includes an instrumentor **154**. Hence, the original HDL description **108** is supplied to the synthesis and place & route system **152**. The synthesis and place & route system **152** can then produce the instrumented design **118** while using not only synthesis and place & route tools but also the instrumentor **154**. In this embodiment, the instrumentor **154** is able to be embedded within synthesis and place & route system **152**. Here, the instrumentor **154** assists with producing the instrumented design **118** which represents the electronic system **104** having the DIC **106** provided within the DUT **102**. However, with the hardware debugging system **150**, the original HDL description **108** need not be modified to produce an instrumented HDL description.

FIG. **2** is a flow diagram of basic instrumentation processing **200** according to one embodiment of the invention. The basic instrumentation processing **200** is, for example, performed by the instrumentor **110** illustrated in FIG. **1**A or the instrumentor **154** illustrated in FIG. **1**B.

The basic instrumentation processing **200** initially receives **202** a HDL description for an electronic system. The HDL description is then analyzed **203** to understand the characteristics of the electronic system. Next, parts (or portions) of the electronic system that are to be examined and/or modified are determined **204**. Typically, the parts of the electronic system to be examined and/or modified (e.g., instrumented) are within a DUT such as the DUT **102** illustrated in FIGS. **1**A and **1**B. Hence, the parts of the electronic system to be examined and/or modified represent various signals and/or components within the DUT. After the parts of the electronic system to be examined and/or modified have been determined **204**, design instrumentation circuitry (DIC) is generated **206**. Preferably, the DIC is determined **204** based on the parts of the electronic system to be examined and/or modified. In this regard, the DIC can be at least partially customized to the application such as the amount or degree of testing or debugging desired. Thereafter, the DIC is incorporated **208** into the electronic system. The DIC can be incorporated **208** into the electronic system (namely, the DUT) in various ways. In one embodiment, the DIC can be incorporated by adding HDL to the original HDL for the electronic system. In another embodiment, the DIC can be incorporated by modifying a netlist description for the electronic system. Following the operation **208**, the basic instrumentation processing **200** is complete and ends.

Design instrumentation (DI) is a process by which a HDL description of an electronic system is analyzed, and then a DIC computed. The DIC is thereafter incorporated (e.g., added) into the electronic system to facilitate debugging. The DIC can be added to the electronic system in a variety of ways. In one embodiment, DIC can be added to the electronic system by adding an HDL description of the DIC to the HDL description of the electronic system. In another embodiment, the DIC can be added to the electronic system during synthesis. The DIC provides mechanisms to control the examination and/or modification of a running electronic system. Thus, the DIC allows to analyze, diagnose, and/or debug the DUT by giving detailed and accurate information about its current state of operation, as well as the state history.

FIG. **3** is a block diagram of an instrumentation system **300** according to one embodiment of the invention. The instrumentation system **300** operates to perform design instrumentation operations to produce an appropriate DIC.

The instrumentation system **300** includes an instrumentor **302**. The instrumentor **302** operates to determine the appropriate DIC for the electronic system (namely, the DUT) that is to be eventually hardware debugged. The instrumentor **302** receives an original HDL description **304** as well as instrumentation directives **306**. The instrumentation directives **306** are instructions to the instrumentor **302** that inform the instrumentor **302** of the portions, parts or areas of the original HDL description **304** that are to be examined and/or modified. The instrumentation directives **306** can be predetermined or interactively provided by a user through a user interface. Additionally, the instrumentor **302** can further receive design constraints **308**, Design For Test (DFT) information **310**, instrumented pre-designed blocks **312** and DIC template(s) **314**.

The design constraints **308** are constraints on the particular design associated with the original HDL description **304**. More particularly, design constraints are limits placed on parameters for an implementation of an electronic system. Some examples of the parameters that can be limited by design constraints include register-to-register propagation delay, system clock frequency, primitive element count, and power consumption. The constraints on the parameters are used by synthesis and place & route tools to guide the implementation of the electronic design.

The DFT information **310** is information about features (e.g., structures) of the original HDL description **304** that pertain to testing. The DFT information is used to facilitate manufacturing testing. For example, the DFT information **310** can provide a description of a scan-chain provided within the original HDL description **304**. The instrumentor **302** can utilize portions of the DFT information **310** to reduce the circuitry required for the DIC.

The DIC can make use of previously instrumented pre-designed blocks **312**. In case the electronic system contains pre-designed blocks which have been instrumented, the DIC can communicate with the previously instrumented pre-designed blocks **312** to facilitate their debugging. The DIC template(s) **314** provide one or more templates for the instrumentor **302** to utilize when producing the DIC.

The instrumentor **302** outputs an instrumented description **316**. In one embodiment, the instrumented description **316** can be represented as an instrumented HDL description in which the original HDL description **304** has been enhanced to include a HDL description of the DIC (see FIG. **1**A). In another embodiment, the instrumented description **316** can represent an instrumented netlist (see FIG. **1**B). The instrumentor **302** also produces an optional DIC HDL description **318**. The DIC HDL description **318** can be utilized by a functional simulator or synthesis and place & route tools. The instrumentor **302** can also produce an optional DIC simulation model **322** that permits functional simulation of the instrumented description **316**. Still further, the instrumentor **302** can output synthesis and place & route constraints **324** and modified DFT information **326**. The synthesis and place & route constraints **324** can be utilized by the synthesis and place & route tools. The modified DFT information **326** can also be used by the synthesis and place & route tools, so that the resulting electronic system is able to be tested as originally designed.

The instrumentation system **300** also includes a design instrumentation database **320** that stores instrumentation information. The instrumentation information includes information on the types of instrumentations that have been done, the DIC and other information as explained in greater detail below. As noted above, an HDL-based hardware debugger (e.g., debugger **122**) eventually utilizes the DIC information stored in the design instrumentation database **320** when performing hardware debugging of the electronic system. Additional details on the design instrumentation database **320** are provided in FIG. **6** below.

FIGS. **4**A and **4**B are flow diagrams of detailed design instrumentation processing **400** according to one embodiment of the invention. The detailed design instrumentation processing **400** is, for example, performed by the instrumentor **110** illustrated in FIG. **1**A, the instrumentor **154** illustrated in FIG. **1**B, or the instrumentor **302** illustrated in FIG. **3**.

The detailed design instrumentation processing **400** initially receives **402** a HDL description of an electronic system. The HDL description is then parsed and analyzed **404**. The analysis **404** of the HDL description can identify portions that cannot be instrumented or that can only be instrumented in certain ways. The result of the analysis **404** can be used to determine whether particular instrumentation directives are valid, and thus can be followed by the instrumentor.

Additionally, one or more of design constraints, DFT information, predetermined instrumentation directives, or pre-designed blocks may also optionally be received **406**. Then, instrumentation directives are determined **408**. Here, instrumentation directives can be predetermined and thus provided or can be determined through user interaction. FIGS. **5**A–**5**D, discussed below, pertain to user interaction to produce instrumentation directives.

After the instrumentation directives are determined **408**, a customized DIC is produced **410** based on the HDL description and the instrumentation directives. Hence, the customized DIC is tailored to the particular HDL description and the particular instrumentation directives. By tailoring the DIC to the particular HDL description and the particular instrumentation directives, the customized DIC makes efficient use of its circuitry. Since the DIC consumes area (e.g., die space) on the hardware product (e.g., semiconductor chip), making the customized DIC efficient and compact is advantageous. In producing the customized DIC, the detailed design instrumentation processing **400** is able to reuse pre-designed blocks that have already been instrumented. In other words, the customized DIC can communicate with existing DICs of pre-designed blocks that represent other portions of the electronic system (or even external systems).

Additionally, the DIC can be optimized **412** to reduce hardware overhead and/or maximize coverage. Here, the optimization **412** to the DIC enables the hardware overhead associated with the DIC to be reduced which is advantageous in producing or using integrated circuit products. For example, cost analysis can be performed during the optimization to explore the different structures in the context of a given implementation technology and given design constraints. Variations of the DIC can thus be explored in order to minimize the overhead of the DIC on the hardware in terms of area, delay, power consumption, routability, and/or testability. Variations of the DIC can be described via DIC templates. The optimization **412** can also try to increase the effects of the instrumentation with regards to the hardware overhead. For example, if some certain signals can be examined, some other signals may also be able to be examined without any or minimal hardware overhead.

Next, a decision **414** determines whether design constraints have been provided. Typically, the design constraints are provided in a file which contains specifications for area, delay, power consumption, routability and testability. When the decision **414** determines that design constraints have been provided, then the DIC may be modified **416** in view of the design constraints. Also, modifications to the design constraints may be performed so that the overall design of the electronic system (including the DIC) complies with the intent of the original design constraints. For example, timing constraints may be changed to reflect the insertion of the DIC. In addition, additional design constraints might be generated, which, for example, may be used to guide synthesis and place & route tools in optimizing the DIC.

Following operation **416**, as well as following the decision **414** when design constraints are not provided, a decision **418** determines whether DFT information has been provided. When the decision **418** determines that DFT information has been provided, then the DFT information is complied with or reused **420**. When complied with, the detailed design instrumentation processing **400** renders the customized DIC compatible or compliant with the DFT information (e.g., existing DFT structures in the design). For example, scan-chains or boundary-scans can be provided or modified to take into account the DIC. Alternatively, when the DFT information is reused, the customized DIC can make use of portions of the circuitry made available through the DFT information and thereby make use of existing circuitry. The modifications to the DFT information can reflect the ability of the DIC to utilize portions of the circuitry within the electronic system associated with the DFT information as well as with the ability to modify the DFT information to preserve the intent of the designer after the DIC is included within the electronic system.

Following the operation **420**, as well as following the decision **418** when the DFT information is not provided, a decision **422** determines whether instrumented, pre-designed blocks have been provided. When the decision **422** determines that instrumented, pre-determined blocks have been provided, then the DIC of each instrumented, pre-designed block is connected **424** to the current DIC. This facilitates debugging of the electronic system which contains pre-designed blocks.

Following operation **424**, as well as following the decision **422** when instrumented, pre-designed blocks are not provided, DIC information is stored **426** to a design instrumentation database. The DIC information includes a description of the DIC, the instrumentation directives, and DIC connectivity information. The DIC information can also include cross-reference data that relates elements in the design of the electronic system (i.e., hardware implementation) to and from the HDL description. Then, the customized DIC can then be added **428** to the electronic system. The customized DIC can be added **428** to the electronic system in a variety of different ways. For example, with respect to an embodiment such as illustrated in FIG. **1**A, the customized DIC can be added **428** to the electronic system by producing the instrumented HDL description which describes the electronic system with the DIC included therein. Alternatively, with respect to an embodiment such as illustrated in FIG. **1**B, the customized DIC can be added to the electronic system by modifying a netlist that defines the electronic system.

Following operation **428** the detailed design instrumentation processing **400** operates to produce and output **430** the instrumented description, an optional DIC simulation model and an optional DIC HDL description. The DIC simulation model can be used by a simulator when functionally simulating the operation of the DUT. The DIC HDL description may for example also be used for simulation. Following the operation **430**, the detailed design instrumentation processing **400** is complete and ends.

As noted above, the instrumentation directives can be predetermined and thus provided or can be determined through user interaction. When the instrumentation directives are predetermined, they can be obtained from a design instrumentation file. In one embodiment, the instrumentation directives specify design visibility, design patching and design control criteria for particular portions of the design for the electronic system.

Design Visibility (DV) is monitoring the entire or partial state of the DUT at, and relative to, predetermined events. An important aspect of DV is relating the states of operation back to identifiers in the original HDL description for examination during HDL-based hardware debugging. In one embodiment, DV is done by sampling the values of one or more signals of the DUT for a particular time interval determined by one or more predetermined events. The events are determined by Design Control which is described below. Design Visibility serves to monitor the state of operation of the DUT, but does not alter the DUT's behavior in any way. However, in some situations, it is advantageous to have a method to alter the behavior of the DUT after the hardware has been fabricated. Design Patching (DP) is to alter the behavior of the DUT to a predetermined particular desired state at predetermined events. The events are determined by Design Control which is described below. A particular desired state of a DUT is a particular setting of the values of all or a subset of all storage components in the DUT.

Design Control (DC) provides the designer with a method to specify the events that control DV and DP. DC can be accomplished by one or more trigger conditions. A trigger condition is a conditional expression comprising HDL identifiers where the conditional expression denotes a combination comprising a particular state and/or state transition, and/or history of states and/or history of state transitions, the DUT, or a portion of it, can be in. Each time a particular trigger condition is met an associated trigger event is produced. One or more trigger events can be combined to issue a particular predetermined trigger action which may control the DV and DP and may control other functions related to HDL-based hardware debugging. A unique combination

comprising one or more units of DV and/or DP all controlled by the same trigger action forms a trigger action group.

A watch-point is a special case of a trigger condition which is explicitly defined using a predetermined conditional expression of HDL identifiers. A watch-point has no direct relationship with the HDL description other than its expression is made up with identifiers of the HDL description.

A break-point is a special case of a trigger condition, where the trigger condition is implicitly specified by selecting a particular source code location in the HDL description. A source code location is a unique combination comprising a file name, a line number and a column position within a textual HDL description.

An error trap is a special case of a watch-point where the trigger condition describes an erroneous or undesired state of the hardware. A property check is a special case of an error trap where the trigger condition is explicitly specified by a particular property of a portion of the hardware. In the event such property is not fulfilled the trigger condition is met. Properties to be checked can either be implicitly derived from the functionality of the hardware or explicitly given by the designer of the electronic system.

FIG. **5**A is a flow diagram of selection processing **500** according to one embodiment of the invention. The selection processing **500** pertains to user interaction with the HDL description to produce instrumentation directives. The selection processing **500** is, for example, performed by operation **406** illustrated in FIG. **4**A when determining instrumentation directives.

The selection processing **500** initially displays **502** a HDL description. The HDL description pertains to the electronic system. At this point, a user can interact with a graphical user interface to make a specific instrumentation directive with respect to the HDL description being displayed. Optionally, to guide a user in his selections, the results of an analysis of the original HDL description can be displayed as well (e.g., operation **404**, FIG. **4**A). Examples of the particular types of instrumentation directives include a selection of a trigger condition, a sampling signal or a patching signal. Hence, a decision **504** determines whether a trigger condition selection has been made. When the decision **504** determines that a trigger condition selection has been made, then trigger condition selection processing **506** is performed. Alternatively, when the decision **504** determines that a trigger condition selection has not been made, then a decision **508** determines whether a sampling signal selection has been made. When the decision **508** determines that a sampling signal selection has been made, then sampling signal selection processing **510** is performed. On the other hand, when the decision **508** determines that a sampling signal selection has not been made, then a decision **512** determines whether a patching signal selection has been made. When the decision **512** determines that a patching signal selection has been made, then patching signal selection processing **514** is performed. Following any of operations **506**, **510** and **514**, as well as following the decision **512** when a patching signal selection has not been made, instrumentation optimization can be performed **516**. The instrumentation optimization operates to consolidate the various selections so that the DIC required to implement the various trigger conditions, sampling signals and patching signals can be efficiently implemented. Following the operation **516**, a decision **518** determines whether more selections are to be made by the user. When the decision **518** determines that more selections are to be made, then the selection processing **500** returns to repeat the decision **504** and subsequent operations.

Alternatively, once the decision **518** determines that no more selections are to be made, the selection processing **500** is complete and ends.

The trigger condition selection processing **506** illustrated in FIG. **5**A can be utilized to select or establish implicit trigger conditions or explicit trigger conditions. An example of an implicit trigger condition is a break-point, and an example of an explicit trigger condition is a watch-point, or an error trap, or a property check.

FIG. **5**B is a flow diagram of break-point processing **520** according to one embodiment of the invention. The break-point processing **520** represents an embodiment of the trigger condition selection processing **506** in the case in which an implicit trigger condition (namely, a break-point) is involved.

The break-point processing **520** initially identifies **522** feasible break-point conditions and types. Typically, such information is obtained analyzing the original HDL description (e.g., operation **404**, FIG. **4**A). Next, the feasible break-point conditions and types are displayed **524**. Here, the feasible break-point conditions and types can be displayed to a user by a user interface. At this point, a user is able to select a location within the HDL description of the electronic system where a break-point is to be set. In one embodiment, a user interface assists the user in making such a location selection with respect to the HDL description (i.e., HDL location). A decision **526** determines whether a HDL location has been selected. When the decision **526** determines that a HDL location selection has not yet been made, then the decision **526** causes the break-point processing **520** to await such a selection. Once the decision **526** determines that a HDL location has been selected, then a decision **528** determines whether the selected HDL location is permitted. In other words, the decision **528** determines whether it is valid to instrument the location within the HDL description of the electronic system with a break-point. When the decision **528** determines that the selected HDL location is not permitted, then an error message is displayed **530**. On the other hand, when the decision **528** determines that the selected HDL location is permitted, then the status type of the selected break-point is updated **532**. Next, break-point information is entered **534** into the trigger condition database for later processing. The break-point information comprises the IDL location of the selected break-point, and the current status type. According to one embodiment, the status type for a selected break-point is "selected".

FIG. **5**C is a flow diagram of explicit trigger condition selection processing **540** according to one embodiment of the invention. As noted previously, one example of an explicit trigger condition is a watch-point. The explicit trigger condition selection processing **540** begins with a decision **542** that determines whether a trigger condition expression has been received. In one embodiment, a user interface assists the user in providing such information. The trigger condition expression defines the explicit trigger condition being set. When the trigger condition expression has not yet been received, the decision **542** causes the explicit trigger condition processing **540** to await receipt of such information (selections). When the decision **542** determines that a trigger condition expression has been received, the status type of the selected trigger condition is updated **544**. For example, the status type for the selected (explicit) trigger condition is "selected". Then trigger condition information is entered **546** into the trigger condition database. The trigger condition information includes the trigger condition expression, the HDL identifiers involved in building the trigger condition expression, and a status type.

Although the break-point processing **520** and the explicit trigger condition processing **540** illustrated in FIGS. **5**B and **5**C pertain to selection and/or entry of trigger conditions, it should be noted that selections can also be made to de-select previously selected trigger conditions. Such processing is generally similar to the selection processing, with the major exception being that the status type of the selected trigger condition is updated to "non_selected", meaning that no instrumentation shall be performed regarding to that portion of the HDL description.

FIG. **5**D is a flow diagram of sampling signal selection processing **560** according to one embodiment of the invention. The sampling signal selection processing **560** is, for example, one representative implementation of the sampling signal selection processing **510** illustrated in FIG. **5**A.

The sampling signal selection processing **560** begins with a decision **562** that determines whether a signal selection has been received. Here, a user is able to select signals by selection of an HDL identifier within the HDL description of the electronic system. In one embodiment, a user interface assists the user in making such a selection with respect to the HDL description. Hence, the decision **562** determines whether such a signal selection has occurred. When the decision **562** determines that a signal selection has not yet occurred, the sampling signal selection processing **560** awaits such a selection. Once the decision **562** determines that a signal selection has been received, then a decision **564** determines whether the selected signal is to be associated with an existing trigger action group of a prior signal selection or whether it becomes a member of a new trigger action group. When decision **564** determines that the signal selection is to be associated with an existing trigger action group, a decision **566** determines whether the user has selected an existing trigger action group. In one embodiment, a user interface assists the user in making such a selection. When the decision **566** determines that a trigger action group selection has not yet been received, the sampling signal selection processing **560** awaits such a selection. Once the decision **566** determines that a trigger action group has been selected, the selected signal is associated **568** with the selected trigger action group. On the other hand, when the decision **564** determines that the selected signal shall become a member of a new trigger action group, a new trigger action group is created **570** and the selected signal is associated **568** with that new trigger action group. Following operation **568**, the status type of the selected signal is updated **572**. The status type for a selected signal is updated **572** to "selected", meaning that the selected signal is selected for instrumentation. Following operation **572** the selected signal is entered **570** into a signal database (see FIG. **6**). Following the operation **570**, the sampling signal selection processing **560** is complete and ends.

Patching signal selection processing can also be performed in a similar manner as the sampling signal selection processing **560** illustrated in FIG. **5**D. In other words, the patching signal selection processing **514** illustrated in FIG. **5**A can also be represented by the processing **560** illustrated in FIG. **5**D. Besides selection of sampling or patching signals in accordance with the processing illustrated in FIG. **5**D, similar processing can also be performed to de-select sampling or patching signals, with the major exception that the status type of the selected signal would be updated to "non_selected", meaning that no instrumentation shall be performed regarding that particular signal.

Design instrumentation databases can be structured in a variety of ways. FIG. **6** is a diagram of a design instrumentation database **600** according to one embodiment of the invention. The design instrumentation database **600** is, for example, suitable for use as the design instrumentation database **114** illustrated in FIGS. **1**A and **1**B or the design instrumentation database **320** illustrated in FIG. **3**.

The design instrumentation database **600** includes a break-point database **602** that stores break-points. The design instrumentation database **600** also includes a signal value database **604** that stores signals within the electronic system that are to be sampled or patched. Hence, the break-points and the signal values, respectively stored in the break-point database **602** and the signal value database **604**, represent instrumentation directives (e.g., design visibility, design patching and/or design control criteria) that govern the characteristics of the resulting DIC and its capabilities. Additionally, the design instrumentation database **600** includes a DIC database **606**, a cross-reference database **612**, and a Register-to-Physical (R2P) database **614**. A representation of the resulting DIC that is produced by the instrumentor is stored in the DIC database **606**. The cross-reference database **612** stores the associations of HDL identifiers (variables) within the HDL description to broaden the design visibility. The R2P database **614** stores translations from registers to physical addresses. The registers are, for example, registers of the DIC used to configure the DIC and hold the status of the DIC and the DUT during hardware debugging. Physical addresses are given for each register to access that register in its implementation inside the DIC. Further, the design instrumentation database **600** includes a text-to-netlist (T2N) database **608** and a netlist-to-text (N2T) database **610**. The T2N database **608** and the N2T database **610** provide for each HDL identifier the associations between the HDL location and elements within the netlist (internal representation of the electronic system).

FIG. **7**A is a block diagram of an instrumentation system **700** according to one embodiment of the invention. The instrumentation system **700** represents a more detailed block diagram of an instrumentor together with a design instrumentation database. For example, the instrumentation system **700** can be a more detailed embodiment of the instrumentation system **300** illustrated in FIG. **3**.

The instrumentation system **700** receives a HDL description **702** of an electronic system. A Design Instrumentation (DI) graphical user interface **704** can display the HDL description on a display device. A user can interact with the graphical user interface **704** to make or enter instrumentation directives. A front-end module **706** receives the HDL description **702** and parses the HDL description **702** to form a parse-tree structure. The resulting parse-tree structure is stored in a parse-tree database **708**. A code generation module **710** reads the parse-tree structure from the parse-tree database **708** and produces a hierarchical design representation associated with the electronic system. The hierarchical design representation provides a description of the designs behavior and structure, such as a hierarchical netlist. The hierarchical design representation is stored in a hierarchical design database **712**. A DI optimization module **714** interacts with the information stored in the hierarchical design database **712**. The information stored in the hierarchical design database **712** is also supplied to an analysis module **716**. The analysis module **716** interacts with the parse-tree database **708** as well as the hierarchical design database **712** to analyze the HDL description of the electronic system design. The analysis includes control flow analysis which determines the feasible break-points which are stored in a trigger condition database **718**. Control flow analysis is further described in "High-Level Synthesis" by Daniel D. Gajski et al., Kluwer Academic Publishers, 1992, which is hereby incorporated by reference. For each location in the HDL description which correlates to a control flow branch condition node, a unique combination of the HDL location and the trigger condition given by the control flow condition can be added as a feasible break-point into the trigger condition database **718**. The purpose of control flow analysis is to reflect that break-points can be set at very particular locations in the HDL description which pertain to HDL control flow statements.

The instrumentation system **700** also includes a location module **724** that interacts with the parse-tree database **708** and the hierarchical design database **712** to produce source code location information represented as T2N information for a T2N database **726** and N2T information for a N2T database **728**. The T2N information provides a method to obtain all elements in the parse-tree database **708** or the hierarchical design database **712** which refer to an identifier at a given location in the HDL description. The N2T information provides a method to relate a given element of the parse-tree database **708** or the hierarchical design database **712** to the originating location in the HDL description. A location query manager **730** interacts with the T2N database **726** and the N2T database **728** to allow a DI manager **732** to relate a location within the HDL description **702** to an element within a netlist (i.e., the parse-tree and/or the hierarchical design representation) and vice versa. The DI manager **732** receives the instrumentation directives, processes them and adds them to the appropriate database (i.e., the trigger condition database **718** or the signal database **722**). Instrumentation directives can be given using file-based DI criteria **734**, interactively by the graphical user interface **704**, or via pragmas in the HDL description. The use of instrumentation directives is explained in greater detail below. The DI manager **732** then interacts with the trigger condition database **718**, the signal database **722**, the location query manager **730**, and the DI optimization module **714** to check each instrumentation directive for its validity. The information regarding the validity is available in the trigger condition database **718** and the signal database **722**.

The DI optimization module **714** receives trigger conditions from the trigger condition database **718** and also receives a DIC template from a DIC template database **720**. Still further, the DI optimization module **714** interacts with a signal database **722** to receive-signals that are to be examined and/or modified. The DI optimization module **714** performs various optimizations regarding the instrumentation directives to reduce the hardware overhead and/or broaden the instrumentation coverage. Additional details on DI optimization are provided below.

For the above-mentioned location determinations with respect to selections, the DI manager **732** queries the location query manager **730** to refer to identifiers in the HDL description **702**, elements in the parse-tree database **708**, and elements in the hierarchical design database **712**.

Selection status types are used to hold the selection information (i.e., the instrumentation directives) and exchange the selection information between the DI user interface **704**, the DI manager **732** and the DI optimization module **714**. The selection status types used for the selection of implicit trigger conditions, explicit trigger conditions, sampling selections and patching selections can comprise: feasible, selected, implied, and not_selected.

The instrumentation directives can be provided in at least three ways, namely, user-based (interactive), file-based, and via pragmas in the HDL description. The user-based approach has been described above. In general, a user (e.g., an electronic system designer) makes design visibility, design patching, and design control selections. More particularly, the designer can select in the HDL description which break-points, watch-points, error-traps, and property checks will be available for activation during HDL-based hardware debugging. These selections are stored in the trigger condition database **718**. The designer also selects in the HDL description which signals shall be available for examination during HDL-based hardware debugging. These selections are stored in the signal database **722**. The designer selects in the HDL description which signals shall be available for patching during HDL-based hardware debugging. These selections are stored in the signal database **722**.

When instrumentation directives are provided in a file, the file-based DI criteria **734** is a human and/or computer readable rule set which describes which signals shall be made visible, which signals shall be made patchable, which break-points are enabled, and which trigger conditions shall be made detectable. The directives in the file-based DI criteria **734** may be expressed in any of the HDL languages that the system accepts as input or may be expressed in a specifically designed language. The directive to select an explicit trigger condition can, for example, comprise a keyword to denote that the selection is a trigger condition, and a conditional expression of HDL identifiers which must be met to issue a trigger event. Implicit trigger conditions, such as break-points, can, for example, be specified by a source code location in the HDL description. The directive to select a signal for sampling can, for example, comprise a keyword to denote that the selection is for a to-be-sampled signal, the unique HDL identifier of the selected signal, and an associated trigger action group. The directive to select a signal for patching can, for example, comprise a keyword to denote that the selection is for a to-be-patched signal, the unique HDL identifier of the selected signal, and an associated trigger action group. The file-based DI criteria **734** can be directly read by the DI manager **732** which stores selections of trigger conditions into the trigger condition database **718** and stores selections of signal values to be made visible and/or patchable into the signal database **722**.

As noted above, the instrumentation directives can be provided via pragmas in the HDL description of an electronic system. Pragmas are HDL code fragments which are inserted into the HDL description to define design visibility, design patching and design control. These pragmas are added to the HDL description such that the behavior of the design of the electronic system is not altered. One implementation adds pragmas to a HDL description as specially-marked HDL comments. By placing the pragmas in comments, other tools which read the HDL description containing the pragmas will be unaffected. However, the front-end module **706** can recognize and interpret these pragmas inside the comments. More particularly, providing instrumentation directives via pragmas can be accomplished by the front-end module **706** recognizing the pragmas enclosed within comments and placing the appropriate information into the parse-tree database **708**. This information is read by the DI manager **732** which stores the necessary information in the trigger condition database **718** and the signal database **722**.

Several examples of pragmas are provided below. These pragmas are written in the form of a HDL comment with an indicator (e.g., "B2SI") to differentiate them from other comments. In the following examples, following the identifier "B2SI", the remainder of the pragma describes either a design control, or a design visibility, or a design patching directive. The exact form of the pragmas depend on the HDL language being used. The following are examples of pragmas written in Verilog HDL.

The following example shows a comment including a pragma for design control.

```
always @( a or b or c or d or e or f ) begin
    if( cond == 4'b1111 ) begin
        // B2SI trigger("trigger_name", (a == 2'b10) &&
(d * e < f + 5'b1100));
        end
    end
```

This pragma produces a trigger condition that is active if the expression

$$(a==2'b10)\&\&(d*e<f+5'b1100)$$

evaluates to true. The expression has the same meaning and variable scoping as it would were it a regular HDL expression. This trigger can also be placed in the control flow of the design so the trigger will not be active unless the control flow is active. In this example, (cond==4'b1111) must also be met to issue a trigger event. The trigger condition has a nane ("trigger name") so that other pragmas may refer to this trigger condition.

The following example shows a comment including a pragma for signal visibility.

```
module mod1( in1, in2, in3, out );
    input in1, in2;
    input in3; // B2SI visible
    output out;
    ...
```

Here, the visibility pragma is being used to mark "in3" as visible.

The following example shows a comment including a pragma for signal patching.

```
module mod2( in1, in2, in3, out );
    input in1, in2;
    input in3;
    output out;
    reg [1:0] aa; // B2SI patchable
```

Here, the patching pragma is being used to mark "aa" as patchable. The trigger condition for the sampling and/or patching can be specified by associating it with a trigger action group (by referring to a trigger name, for example "trigger_name"), or during HDL-based hardware debugging.

The optimization of the design instrumentation can enhance its effects and can reduce hardware costs of the DIC. One example of an optimization for enhancing the effects of the instrumentation is implication analysis. One example for an optimization which aims to reduce the hardware costs of the DIC is resource sharing.

The selections of various trigger conditions and signals for sampling and/or patching may potentially imply other signal selections based on their controlability and observability dependencies. Controlability and observability are, for example, commonly used concepts in Automatic Test Pattern generation of combinational and sequential logic. See D. Bhattacharya and J. P. Hayes, "Hierarchical Modeling for VLSI Circuit Testing," Boston: Kluwer, 1990, p. 159, which is hereby incorporated by reference. Implication analysis works as follows. Initially, the hierarchical design database **712** and the DI optimimization module **714** are consulted to determine whether a trigger condition with the status type ""selected" implies certain other trigger conditions. If so, the implied trigger conditions can also be detected during HDL-based hardware debugging, have their status type set to "implied", and be stored into the trigger condition database **718**. Secondly, the hierarchical design database **712** and the DI optimization module **714** can be consulted to determine whether certain other signal values are implied by the selected signals. In particular, the implied signals can be derived from the selected signals plus some calculations during HDL-based hardware debugging. Each implied signal is then stored with its status type set to "implied" into the signal database **722**.

Resource sharing is a widely used optimization which is, for example, used in synthesis. Although resource sharing can be performed using many different approaches, in one approach to resource sharing, the DI optimization module **714** operates to share resources in the DIC as follows. First, by consulting the DIC template database **720**, the DI optimization module **714** knows about the structure and the cost model of the DIC and can determine whether trigger conditions and signals to be sampled have commonalities which can be utilized for resource sharing. Second, the hierarchical design database **712** and the DIC template database **720** can be consulted by the DI optimization module **714** when determining whether other signals should instead be sampled, since such signals imply all the selected signals, but their sampling requires less hardware overhead or leads to additional signal visibility. Third, by consulting the DIC template database **720**, the DI optimization module **714** knows about the structure and the cost model of the DIC and can determine whether trigger conditions and signals to be sampled have commonalities which can be utilized for resource sharing. Fourth, by consulting the DIC template database **720**, the DI optimization module **714** knows about the structure and the cost model of the DIC and can determine whether signals to be patched have commonalities which can be utilized for resource sharing.

Once the trigger conditions and the signals to be sampled and/or patched are determined, other portions of the HDL design can be integrated even if such portions are not described by a synthesizable HDL description but are available as synthesized and physically realized hard blocks, such as previously designed hard blocks. If the hard blocks are synthesized from instrumented HDL and include DIC, regardless whether the DIC is a complete or a partial, the previously inserted DIC can be re-used for debugging the hard blocks. The distinction between partial versus complete DIC is described in greater detail below.

In order for a hard block to be re-used, it should have associated DI data stored in an associated design instrumentation database. FIG. 7B is a diagram of a hard block resolution system **750** according to one embodiment of the invention. The data needed are a hard block's DIC database **752**, a hard block's trigger condition database **754**, a hard block's signal database **756**, and optionally HDL description **758**. Often, vendors of hard blocks do not want to expose the internal workings of their design by showing its HDL description (e.g., source code). To accommodate this need, the HDL description is not required to describe the entire

hard block's functionality. Some minimal HDL description providing just enough text to examine signals, to set watch-point expressions for the signals, and to set break-points at HDL locations which refer to implemented trigger detection circuitry is enough to enable HDL-based hardware debugging of the hard blocks. For example, a hard block implementing a simple controller might expose the controller state variable for sampling and for triggering on its value. It might also allow a user to set a break-point when the machine makes certain transitions or receives certain signals from the circuitry to which it is connected. A hierarchy and hard block resolver **760** processes the information from the hard block's DIC database **752**, the hard block's trigger condition database **754**, the hard block's signal database **756** and the optional HDL description **758**, and merges same into the current HDL design's DIC database **736**, the trigger condition database **718**, the signal database **722**, and the original HDL description **702**. As a result, the resolved information will be available during HDL-based hardware debugging.

The instrumentor **700** can also perform cross-reference analysis to gather and store data in the design instrumentation phase such that the HDL-based hardware debugger will be capable of examining signals in the HDL description. Additionally, if the design instrumentation optimization determines that other signals could be derived from the sampled signals, the HDL-based hardware debugger needs the HDL expressions to compute the derived signals "on the fly" from the sampled signals. The expressions are calculated during cross-reference analysis and stored in the cross-reference database **1504**.

FIG. **15** is a block diagram of a portion of an instrumentation system **1500** which includes a cross-reference analysis module **1502** and a cross-reference database **1504** according to one embodiment of the invention. The cross-reference analysis module **1502** can be provided within the instrumentation system **700**, and the cross-reference database **1504** can be provided within the design instrumentation database **612** and utilized by the instrumentation system **700**. The cross-reference analysis module **1502** can couple to the location query manager **730**, the hierarchical design database **712** and the signal database **722**. The cross-reference analysis module **1502** reads signal information from the signal database **722**. Each entry in the signal database **722** corresponds to one signal that is either selected or implied to be made visible. Each entry in the signal database **722** also comprises information on whether the signal is to be sampled and/or patched in the DIC or whether the signal is derived from other to-be-sampled signals. In one embodiment, for each signal that is derived from other to-be-sampled signals, the following operations are performed. First, the cross-reference analysis module **1502** queries the HDL location information of the signal from the location query manager **730**. The cross-reference analysis module **1502** looks up the signal in the hierarchical design database **712** and determines the proper HDL expression to compute the derived signal from the set of sampled signals. The cross-reference analysis module **1502** then writes the HDL expression into the cross-reference database **1504**.

The instrumentor **700** can also perform Design-for-Test (DFT) analysis. If the electronic system contains additional circuitry for testability such as scan-chains, boundary scan logic, JTAG tap-controllers or similar DFT features, and if such circuitry is described in the DFT information (file) **310**, then the circuitry can be shared to reduce the hardware overhead of the DIC. Example formats of such a DFT information file is the Boundary-Scan Description Language (BSDL) or Hierarchical Scan Description Language

(HSDL), both defined by the IEEE 1149.1 JTAG standard available from the Institute of Electrical and Electronic Engineers (IEEE) in Piscataway, N.J., which is hereby incorporated by reference.

FIG. **16** is a block diagram of a portion of an instrumentation system **1600** which includes a DFT analysis module **1602** according to one embodiment of the invention. The DFT analysis module **1602** receives information about the DFT information **310**, the current implementation of the DIC as stored in the DIC database **736** and the hierarchical design database **712**, and the register-to-physical (R2P) address translation information (e.g., table) provided in the R2P database **614**. The result produced by the DFT analysis module **1602** is the modified DFT information **326**, namely, altered register-to-physical address translation information (e.g., table), which is needed by post-processing DFT tools. The R2P database **614** needs to be updated each time DIC registers have been moved to different physical locations.

FIG. **17** is a data flow diagram illustrating DIC creation processing **1700** according to one embodiment of the invention. The DIC and the instrumented design is created at the end of the design instrumentation process. The DIC is described by the DIC HDL description **318**. The instrumented design is described by the instrumented HDL description **316**. Additionally, various components of the design instrumentation database **600** are established, including the R2P database **614**, the DIC database **736**, the signal value database **604**, and the break-point database **602**. The DIC creation processing **1700** has a data flow described as follows.

First, the trigger condition database **718** and the signal database **722** (which can result from the DI manager **732**) are processed by a trigger condition code generation module **1706** and a signal code generation module **1708**, respectively.

Second, for each entry in the trigger condition database **718**, the trigger condition code generation module **1706** generates the structures of the trigger detection circuitry for the DIC according to the DIC template database **720**. Then, such structures are added to the hierarchical design database **712**. In addition, proper DIC register configuration rules can be added to a DI rule database **1710**.

Third, for each signal designated as to-be-sampled in the signal database **722**, the signal code generation module **1708** creates circuitry to sample such signal according to the structure in the DIC template database **720**, and adds the structures to the hierarchical design database **712** and the proper DIC register configuration rules to the DI rules database **1710**.

Fourth, for each signal designated as to-be-patched in the signal database **722**, the signal code generation module **1708** generates the circuitry to patch such signal according to the structure in the DIC template database **720**, and adds such structures to the hierarchical design database **712** and the proper DIC register configuration rules to the DI rule database **1710**.

Fifth, a break-point analysis module **1712** then reads the trigger detection circuitry from the hierarchical design database **712** and the register configuration rules from the DI rule database **1710**. Knowing the structure of the DIC from the DIC template database **720**, the break-point analysis module **1712** creates the break-point database **602**. The break-point database **602** comprises all the rules for which the location break-points are possible to be set. The break-point database **602** also comprises rules about mutual exclusivities between break-points due to hardware restrictions in the DIC. For example, a certain break-point may not be used with another

break-point because both break-points require the same hardware resource in the DIC.

Sixth, signal analysis module **1714** then reads the signal sampling/patching circuitry from the hierarchical design database **712** and the register configuration rules from the DI rule database **1710**, and knowing the structure of the DIC from the DIC template database **720**, the signal value database **604** is created. The signal value database **604** comprises all the rules about mutual exclusivities between signal values for sampling and/or patching due to hardware restrictions in the DIC.

Seventh, the DIC generation module **1716** then reads the DI rule database **1710** and the DIC template database **720** and connects all trigger detection circuitry and all signal sampling/patching circuitry to a trigger processing unit (TPU)(see FIG. **8**). Also, the configuration and the status registers, and the communication controller are added and connected. The complete structure of the DIC is then written to the hierarchical design database **712** and the entire and complete rule set to configure the registers of the DIC is written to the DIC database **736**.

Eighth, a DIC register-to-physical mapping module **1718** maps each register configuration and each status register in the DIC into an address space of physical memory in the design to produce the R2P database **614**. For example, the physical memory could be implemented as a set of scan-chains, in which case the physical address of a configuration or status register would be given by the index of the scan-chain used and the bit position within the scan-chain.

Ninth, a DIC writer module **1720** produces the synthesizable HDL description of the DIC (e.g., DIC HDL description **318**), defined by the configuration and status information in the DIC database **736** and the DIC structure stored in the hierarchical design database **712**.

Tenth, the DIC writer module **1720** also reads in the original HDL description **304**, annotates it with the information about the DIC from the hierarchical design database **712** and the DIC database **736**, and writes out the instrumented HDL description **316** (e.g., annotated HDL source code) of the instrumented design for further processing by synthesis and place-and-route tools.

Eleventh, to support regression testing of the instrumented design using functional simulation, the optional DIC simulation model **322**, including the necessary HDL wrapper files, is written by a DIC simulation model generation module **1722**.

Twelfth, a design constraint analysis module **1724** reads the design constraint file **308** which holds all constraints created by the designer. The design constraint analysis module **1724** then adjusts the original set of constraints to produce the instrumented design constraint file **324** for the instrumented design.Design constraint analysis is described in greater detail below.

Annotating the HDL description adds the HDL description of the DIC to the original HDL description and connects the DIC to the portions of the original HDL description for which design visibility, design patching, and design control has been selected. The annotation can be performed automatically. The result of the annotation is the instrumented HDL description. The instrumented HDL description is the original HDL description together with a small amount of HDL description added for the DIC. The annotations may be added to the hierarchical original HDL description in two ways: distributed or monolithic. Distributed annotations are added to each hierarchical element of the original HDL description. Monolithic annotations are added to the top-level element of the HDL design and then connect to other parts of the design. Since distributed annotations are more powerful and more complex than monolithic annotations, distributed annotations will be described in detail below.

A HDL description can be composed of one or more HDL Building Blocks (BBs). Similarly, the DIC is composed of one or more specially-tailored HDL BBs, the DICBBs. One such DICBB can be inserted into each BB in the original HDL description. The BB in the original HDL design is termed the DICBB's host BB (HBB). An example provided below is a Verilog description of a simple building block which consists of some simple logic.

```
module mod3( in1, in2, out );
  input in1, in2;
  output out;
  assign out = ( in1 > in2 );
  endmodule
```

Another example provided below is a Verilog description of the Host Building Block (HBB) above following annotation (i.e., instrumented building block) to include one of the DICBBs with some simple building blocks which consist of one HBB and some simple logic. In the Verilog language the DICBB is an instantiation of a specially-tailored DIC Verilog module.

```
module mod3( in1, in2, out );
input in1, in2;
output out;
assign out = ( in1 > in2 );
DIC__mod1 DIC__instance( in1, in2 );
endmodule
module DIC__mod1( in1, in2 );
input in1, in2;
// specially-tailored DIC goes here
endmodule
```

Each DICBB communicates with its associated HBB by connecting to the HBB's signals. Design visibility of a particular HDL identifier residing in a HBB can be accomplished by connecting the identifier to the associated DICBB. The internal circuitry of the DICBB is created using the knowledge of the signal connections. This mechanism allows design visibility, design patching, and design control to be supported by the DIC. The above example shows a DICBB connected to two HDL identifiers "in1" and "in2". The circuitry inside DIC_mod1 can utilize the signals for the purpose of design visibility of one or both the signals and/or for creating watch-points which monitor one or both of the signals.

If a symbolically-encoded HDL identifier is made visible, symbolic values can be displayed for it during HDL-based hardware debugging. To do this, each symbolic value needs to be associated with the actual binary code assigned to it during synthesis (**116** in FIG. **1**A.). Since it is desirable for the instrumentation to be independent of the synthesis, the HDL-based hardware debugger cannot rely on any information from the synthesis about the association between binary codes and symbolic values. Consequently, each of the symbolic values must be connected to the DICBB so that the circuitry inside the DICBB can explicitly know the binary codes assigned to each symbolic value. During HDL-based hardware debugging, the encoding information is obtained from the instrumented HDL design.

Break-points are supported by adding signals to the HBB which are active when the control flow which the break-point is modeling is active, and are inactive otherwise. The added signals are then connected to the DIC associated with the HBB and are used when the circuitry of the DIC is created. The following example shows the Verilog HDL fragment of a HBB which has simple control flow logic.

```
1   module mod4( in1, in2, out );
2   input in1, in2;
3   output out;
4
5   always@ ( in1 or in2 ) begin
6     if (( in1 == 1'b0 ) || ( in2 == 1'b1 )) begin
7       out = 1'b1;
8     end else begin
9       out = 1'b0;
10    end
11  end
12
13  endmodule
```

Line numbers have been added to the above example for reference purposes, the line numbers are not part of the Verilog description. There are two lines, line 6 and line 8, which can have a break-point. These lines correspond to the two control flow branches which arise from the "if" conditional statement on line 6.

The next example shows the Verilog HDL fragment of the above example annotated such that the added circuitry supports two break-points.

```
module mod4( in1, in2, out );
  input in1, in2;
  output out;
  reg bp1, bp2; // Added during instrumentation
  always@ ( in1 or in2 ) begin
    bp1 = 1'b0;
    bp2 = 1'b0;
    if (( in1 == 1'b0 ) || ( in2 == 1'b1 )) begin
      out = 1'b1;
      bp1 = 1'b1;
    end else begin
      out = 1'b0;
      bp2 = 1'b1;
    end
  end
  DIC__mod2 DIC__instance( bp1, bp2 );
  endmodule
  module DIC__mod2( bp1, bp2 );
  input bp1, bp2;
  // specially-tailored DIC goes here
endmodule
```

Note signals "bp1" and "bp2" have been added to the HBB. Each signal is active (set to logical **1**) only when the control flow branch that the signal is modeling is active. The signals are connected to the associated DICBB DIC_mod2 and can be used by the circuitry inside the DICBB to create break-point circuitry.

The DICBBs in the instrumented HDL design communicate with each other by connecting to identifiers that have been added to their respective HBBs and which are also connected to the HBB's ports. The following example shows the Verilog HDL fragment which consists of two BBs. BB mod6 is instantiated by BB.

```
module mod5( in1, in2, in3, out );
input in1, in2, in3;
output out;
wire tmp_out;
assign out = ( in1 > tmp_out );
mod6 instance( in2, in3, tmp_out );
endmodule
module mod6( com1, com2, out );
input com1, com2;
output out;
assign out = com1 ^ com2;
endmodule
```

The following example shows the Verilog HDL fragment of the above example after being annotated.

```
module mod5( in1, in2, in3, out );
input in1, in2, in3;
output out;
wire tmp_out;
wire DIC_com2; // Added during instrumentation
assign out = ( in1 > tmp_out );
mod6 instance( in2, in3, tmp_out, DIC_com2 );
DIC_mod3 DIC_inst3 ( DIC_com2 );
endmodule
module mod6( com1, com2, out, DIC_com1 );
input com1, com2;
output out;
inout DIC_com1; // Added during instrumentation
assign out = com1 ^ com2;
DIC_mod4 DIC_inst4 ( DIC_com1 );
endmodule
```

The annotation consists of: (1) DICBBs DIC_mod3 and DIC_mod4 which have been added to their respective HBBs mod5 and mod6. (2) Signal DIC_com1 which has been added to HBB mod6, added to the port list of HBB mod6, and connected to DIC_mod4. (3) Signal DIC_com2 which has been added to the HBB mod5 and connected to the DIC_com1 port of the DIC_mod4 DICBB and to the DIC-_mod3 DICBB. Consequently, the DIC_mod4 DICBB communicates with the DIC_mod3 DICBB via the connection of DIC_mod4 to signal DIC_com1 which is connected through port DIC-com1 of mod6 to signal DIC_com2 of mod5 which is connected to DIC_mod3.

An original design of the electronic system (e.g., original HDL description) can be instrumented with either a complete DIC or a partial DIC. A complete DIC comprises a communication controller and a trigger processing unit (TPU). While a complete DIC, such as shown in FIG. **8**, includes a communication controller and a TPU, a partial DIC does not include these components. An original HDL design may be instrumented with a partial DIC if it is to be used inside another instrumented HDL design which has a complete DIC. For example, an original HDL description could be instrumented with a partial DIC if it were to be used as a hard block. Although an instrumented HDL design with a complete DIC can be used as a hard block if its communication controller and TPU are disabled, this wastes hardware and thus space.

Instrumenting with a complete DIC can be accomplished by adding a special DICBB which is referred to as the "master" DICBB (MDICBB) which comprises a communication controller and a TPU. The MDICBB is placed into an HBB of the original HDL design which allows the MDICBB to communicate with the host communication controller. For

example, in a Verilog design, the HBB of the MDICBB would be the Verilog module which is the top-level module in the design hierarchy—the HBB would be the one module in the design which is not instantiated in the Verilog design. The MDICBB is connected to the DICBB in the MDICBB's HBB. Consequently, the MDICBB can communicate with all other DICBBs in the instrumented HDL design so that said MDICBB can gather, process, and transmit to the host communication controller information from the other DICBBs. The following example shows the Verilog HDL fragment of an above example for a basic building block (re module mod3) after it has been annotated.

```
module mod7( in1, in2, out, DIC_com3 );
input in1, in2;
output out;
inout DIC_com3; // Added during instrumentation
assign out = ( in1 > in2 );
DIC_mod5 MDICBB_inst ( DIC_com3 );
endmodule
```

Note that in this example, mod7 is the top-level module of the original HDL design and DIC_mod5 is the MDICBB. DIC_mod5 communicates to the environment by connecting with signal DIC_com3 which has also been made a port of the HBB mod7.

In performing design constraint analysis, the design constraint analysis module **1724** reads the design constraint file **308** which holds all constraints that ensure the HDL design meets the area, delay, power consumption, routability, and/or testability specifications made by the designer of the electronic system. The design constraint analysis module **1724** then analyzes the instrumented HDL design stored in the hierarchical design database **712** and adjusts the original set of constraints to the inserted DIC and possibly adds additional constraints. Both sets of the constraints together can be written into the instrumented design constraint file **324** for the instrumented HDL design. The additional constraints attempt to minimize the impact of the DIC on the area, delay, power consumption, routability, and/or testability of the HDL design.

FIG. **8** is a block diagram of a representative design instrumentation circuit (DIC) according to one embodiment of the invention. The representative DIC **800** includes a plurality of probe circuits, namely probe circuitry **802**, probe circuitry **804** and probe circuitry **806**. The probe circuitry **802**–**806** couple to a trigger processing unit **808**. The trigger processing unit **808** is configurable circuitry which is used to process trigger events and issue corresponding trigger actions. Such correspondance between the trigger events and the trigger actions can be given as complex trigger conditions. A complex trigger condition can be a complex conditional expression between two or more trigger events. Propositional or temporal logic may be used to describe such expressions. The trigger processing unit **808** controls the ability of the DIC **800** to detect trigger conditions and to sample and/or patch signal values. The acts of detection, sampling and patching can be independent from each other. When trigger conditions are detected, the trigger processing unit **808** triggers sampling (visibility) or patching of signals within the DUT. In this regard, the probe circuitry **802**–**806** couple to electrical signals within the DUT. Each of the probe circuitry **802**–**806** is designed to perform a sampling of a signal, a modification to a signal, or a detection of a trigger condition. Typically, these signals or conditions are digital conditions. However, in the case in which the DUT includes analog and digital portions, the probe circuitry **802** can include an analog-to-digital (A/D) converter **810** so as to convert analog signals to digital signals prior to being received at the probe circuitry **802**. The representative DIC **800** also includes status registers **812** and configuration registers **814**. The status registers **812** store certain status information and the configuration registers **814** store certain configuration information.

A communication controller **816** couples to the status registers **812** and the configuration registers **814**. Hence, a HDL-based hardware debugger is able to communicate with the DIC via the communication controller **816**. More particularly, the HDL-based hardware debugger can read and set registers within the status registers **812** as well as within the configuration registers **814**. As a result, the communication controller **816** allows configuration data to be sent to the DIC **800** and status data to be retrieved from the DIC **800**. The communication controller **816** can implement a method (i.e., run-time method) for externally reading and writing the configuration registers **814** which configure the DIC **800** and externally reading the status registers **812** (memory) which store the sample values. In one embodiment, the register values can be read or set using a standard connection defined by the IEEE 1149.1 JTAG standard, available from the Institute of Electrical and Electronic Engineers in Piscataway, N.J., which is hereby incorporated by reference.

In order to maintain flexibility in HDL-based hardware debugging, the DIC is configurable at run-time. Externally configurable registers are used to change the detection of HDL-based trigger conditions and the selection of signals to be sampled and/or patched without the need to re-implement the design of the electronic system.

There is also a general need for the DIC to communicate with components which are not instrumented. This external communication can be implemented by connecting signals between the DIC and the other components. One example would be an external signal that the DIC activates when any trigger condition is met. In another example, the DIC has external connections to notify and be notified about certain conditions which occur in an optional embedded processing unit (e.g., CPU) and thus support hardware/software co-debugging.

Additional details concerning representative implementations for the trigger processing unit **808** and the probe circuitry **802**–**806** are provided below. This circuitry is added to the original design of the electronic system. For the purposes of the discussion below, it is assumed that the hardware debugging system **100** of FIG. **1A** is being used. Hence, the circuitry for the DIC is added to the original HDL description as additional HDL by the instrumentor **110** in producing the instrumented HDL description **112**.

FIG. **9** describes a representative generic configurable circuitry **900** which can implement design sampling and design patching according to one embodiment of the invention. The circuitry **900** includes a register **902**, a multiplexer **904**, a tri-state register **906**, and a storage **908**. When the register **902** is to be sampled, a selector signal **910** selects a register input **912** to drive the register **902** via multiplexor **904**. A sample enable signal **914** enables the tri-state buffer **906** to drive a register output **916** onto a data bus **918**. The storage **908** couples to the data bus **918** and can thus store the value at the register output **916**. For each successive sample, the value on an address bus **920** is incremented. Alternatively, when the circuitry **900** is to be patched, the address bus **920** selects the proper patch value from the

storage **908**. The multiplexor selector signal **910** selects the data bus **918** to drive the input to the register **902** via the multiplexor **904**, and the selector signal **914** disables the tri-state buffer **906**, thereby driving the value from the storage **908** into the register **902**.

Storage **908** can also be implemented by sampling circuitry. Sampling circuitry can use sets of registers or Random Access Memory (RAM) as storage for sampling predetermined signals. The sampled values can thereafter be read from the storage and communicated to the HDL-based hardware debugger. One implementation of storage **908** is a circular buffer of depth M which continuously samples predetermined signals. When a predetermined trigger action occurs, sampling is stopped. At which point the circular buffer contains the M last values of all sampled signals. To save circuitry, the sampling circuitry can be shared for many signals. For example, a configurable crossbar, implemented either as a full crossbar or as a multiplexor network, will allow many signals to share the same storage (e.g., circular buffer).

Design patching can also be implemented by patching circuitry. According to one embodiment, the patching circuitry provides a method for patching predetermined internal signal registers. For each register in the design of the electronic system which is to be made patchable, the patching circuitry can include a companion register and simple control circuitry. The companion register holds the patch value(s) and is run-time configurable. The patching circuitry operates as follows: First, during configuration of the DIC, the companion storage is loaded with a desired value. Second, under the control of a particular trigger action, the patching circuitry forces the patched register to take some configured value from the companion storage. This patching circuitry thus allows patching to be used for many applications including, but not limited to, debugging and fixing previously fabricated hardware.

Design visibility and design patching are controlled by particular trigger actions which are determined by design control circuitry. FIG. **10** illustrates a representative generic configurable trigger detection circuit **1000** according to one embodiment of the invention. The trigger detection circuit **1000** operates to detect trigger conditions and issue trigger events.

The trigger detection circuit **1000** includes a configurable trigger register (TR) **1002** that stores a trigger value that is compared to a monitored signal (ISR) **1004** by a comparator **1006**. The mode of the comparator **1006** can be controlled by a configurable trigger comparison register (TCR) **1008**. Examples of different comparison modes are test for equivalence, test for smaller-than, etc. The ability to configure the trigger register (TR) **1002** and the trigger comparison register (TCR) **1008** allows the electronic system designer the flexability to check for a wide variety of trigger conditions during HDL-based hardware debugging. A configurable trigger enable register (TER) **1010** allows the trigger condition to be activated or disabled. If the trigger condition implemented by comparing the monitored signal (ISR) **1004** to the trigger register (TR) **1002** is met and the trigger enable register (TER) **1010** is enabled, a trigger condition signal **1012** becomes active to denote a trigger event. A trigger detected register (TDR) **1014** can be used to store such a trigger event, which can be subsequently read during HDL-based hardware debugging to determine whether a trigger event has occurred.

While FIG. **10** illustrates the representative generic configurable trigger detection circuit **1000**, for various more specific situations, specialized design control circuitry provides more efficient hardware. Examples of these specific situations, including state based Finite State Machines (FSMs), transition based FSMs, data-path registers, and temporal logic, are described below.

State based FSM design control circuitry provides a configurable method to detect whether an FSM is in a particular state—a condition which depends on the value of the FSM's state register. For simplicity, a one-hot encoded state-machine is described herein. For other state encodings, the design control circuitry can be implemented similarly. FIG. **11** illustrates a representative state based FSM design control circuit **1100** according to one embodiment of the invention. For each FSM state register that is to be instrumented to detect particular states, the state based FSM design control circuit **1100** is added. A to-be-instrumented one-hot encoded FSM **1102** has a state register **1104** which is n bits wide and which is sensitive to the clock signal **1106**. The state based FSM design control circuit **1100** that is added includes a trigger register **1110** which has the same bit-width n as the state register **1104** and which is sensitive to the same clock signal **1106**. An output **1112** of the state register **1104** is compared to an output **1114** of the trigger register **1110** using a combinatorial network **1116**. The combinatorial network **1116** implements a trigger condition signal **1118**. The trigger condition signal **1118** produced by the state based FSM design control circuit **1100** can be a single bit output function and can be described in its behavior by the following Verilog code:

```
module m1116 (n1112, n1114, n1118);
    parameter n = 32;
    input     [n-1:0]   n1112;
    input     [n-1:0]   n1112;
    output              n1118;
    wire                n1118 = | (n1112 & n1114);
endmodule
```

Thus to detect a particular current state in the one-hot encoded FSM **1102**, one can set the corresponding bit in the trigger register **1110** to logical "1". The trigger register **1110** can be configured with appropriate values through a connection (link) **1120**. The trigger condition signal **1118** will then be logically "1" to denote the trigger event.

Transition based FSM design control circuitry provides a configurable method to detect whether a FSM is undergoing a particular state transition—a condition which depends on the value of the state register and also on the activity and values of the input signals of the FSM. For simplicity, a one-hot encoded state-machine is described herein. For other state encodings, the design control circuitry can be implemented similarly.

FIG. **12** illustrates a representative transition based FSM design control circuit **1200** according to one embodiment of the invention. For each FSM that is to be instrumented for detecting particular state transitions, the transition based FSM design control circuit **1200** is added. The to-be-instrumented one-hot encoded FSM **1202** has a state register **1204** which is n bits wide and which is sensitive to a clock signal **1206**. The transition based FSM design control circuit **1200** that is added includes a trigger register **1208** which is sensitive to the clock signal **1206**, and is ○ bits wide where ○ is the number of different state transitions of the FSM **1202**. A combinatorial network **1210** performs a unique one-hot encoding of each different state transition into output **1212** and thus is connected to the n bit wide output

1214 of the state register **1204** as well as to the m bit wide input **1214** of the FSM **1202**. A combinatorial network **1216** is connected to a ○ bit wide output **1218** of the trigger register **1208** and the ○ bit wide output **1212** of the combinatorial network **1210**. A trigger condition signal **1220** is the single bit output of the combinatorial network **1216** and can be described in its behavior by the following Verilog code:

```
module m1216 (n1218, n1212, n1220);
    parameter o = 32;
    input     [o-1:0]   n1218;
    input     [o-1:0]   n1212;
    output              n1220;
    wire                n1220 = | (n1218 & n1212);
endmodule
```

Thus to detect a particular state transition in the one-hot encoded FSM **1202**, the bit in the trigger register **1208** corresponding to the one-hot code of the particular state transition must be set to logical "1". A ○ bit wide connection **1222** can be used to configure the trigger register **1208** with appropriate values. The trigger condition signal **1220** becomes a logical "1" whenever a state transition is active, which denotes the trigger event.

For data-path registers, data-path register design control circuitry provides a configurable method to detect whether a data-path register has a particular current value, whether a data-path register has a particular relationship to other values, or whether a data-path register has just changed its value. FIG. **13** illustrates a representative data-path register design control circuit **1300** according to one embodiment of the invention. The data-path register design control circuit **1300** is coupled to a data-path register **1302** which is sensitive to a clock signal **1304** and which latches the n bit wide input net **1306** into a n bit wide output net **1308**. The data-path register design control circuit **1300** includes one or more of n+1 bit wide trigger registers **1310**, **1312**, **1314** which all are sensitive to the clock signal **1304**. The n bit wide output **1308** of the data-path register **1302** and all the n+1 bit wide outputs **1316**, **1318**, **1320** of the trigger registers **1310**, **1312**, **1314** are then connected as inputs to a combinatorial network **1322**. The combinatorial network **1322** provides configurable pair-wise checking relations between the current value of the data-path register **1302** and the n least significant bits of one of the trigger registers **1310**, **1312**, **1314**. The relation being checked for can be the equality, non-equality, less than, greater than, etc., and such relation can be determined by the user. The most significant bit within each of the n+1 bit wide trigger registers **1310**, **1312**, **1314** is used for enabling (if the bit is set to "1") or disabling (if the bit is set to "0") the checking of the relation and can be described in its behavior by the following Verilog code:

```
module m1322 (n1308, n1316, n1318, n1320, n1324);
  parameter n = 32;
    input     [n-1:0]   n1308;
    input     [n :0]    n1316;
    input     [n :0]    n1318;
    input     [n :0]    n1320;
    output              n1324;
    wire   check0   =   n1316[n]   &   compare0(n12190,
```

-continued

```
n1316[n−1:0]);
        wire   check1   =  n1318[n]   &   compare1(n12190,
n1318[n−1:0]);
        wire   check2   =  n1320[n]   &   compare2(n12190,
n1320[n−1:0]);
        wire              n1324   =   check0   |   check1   |
check2;
    endmodule
```

If one of the relations is satisfied, the trigger condition signal **1324** becomes logical "1" to denote a trigger event.

Temporal logic is an extension of conventional propositional logic which incorporates special operators that operate with time as a variable. Using temporal logic, one can specify how functions behave as time progresses. In particular, temporal logic statements can make assertions about values and relationships in the past, present, and the future. A subset of temporal logic can be used to describe interdependencies between trigger events over a certain time period relative to a given event, at one or more cycles, or for trigger events of the past. FIG. **14** illustrates a representative design control circuit **1400** according to one embodiment of the invention that can be used to implement temporal logic needed for relationships which include signals or trigger events from previous clock cycles. The trigger condition signal **1402** can be delayed by a configurable number of cycles of clock **1404** using delay registers **1406**, **1408** and **1410**. A multiplexor **1412**, under the control of a trigger control register (TCR) **1414**, selects which current or previous value of the signal **1402** is sent to output **1416**. The output **1416** can be used as an input to temporal logic equations.

The selection of the signal to drive the clock input **1404** of the delay registers **1406**, **1408** and **1410** offers powerful functionality as follows. First, when one of the system clock signals is connected to the clock input **1404** of the delay registers **1406**, **1408** and **1410**, events can be delayed relative to the system clock. Second, when a particular trigger condition signal is connected to the clock input **1404** of the delay registers **1406**, **1408** and **1410**, the signal **1402** is delayed relative to the trigger condition signal.

To implement the capability to control the processing of particular trigger events over relatively longer periods of time, counters can be used. The counters operate by loading a configured value, counting down from the loaded value to zero, and then issuing an event when zero is reached. The selection of the signal to drive the clock input of the counter offers powerful functionality. First, when one of the system clock signals is connected to the clock input of the counter, trigger events can be delayed relative to the system clock. Hence, trigger events can be made to depend on the system time. For example, trigger events might be enabled for a certain time period and become disabled otherwise, or become enabled after some time period. Second, when a particular trigger condition signal is connected to the clock input of the counter registers, the operation of the counter is dependent on the trigger condition signal.

As noted previously, a DIC includes a trigger processing unit (TPU) to process all incoming trigger events and to issue appropriate outgoing trigger actions based on the incoming trigger events. The TPU provides a configurable method for calculating complex trigger combinations and other relationships between one or more of the trigger events to produce the trigger actions. The trigger events for processing by the TPU are the trigger condition signals of the design control circuitry of the DIC as described above or signals from circuitry external to the DIC. For example, in hardware/software co-debugging of embedded CPUs, such external signals may be the error signals of the CPUs. In another example, when multiple DICs are coupled (e.g., daisy-chained) to support debugging of multi-chip systems, another such trigger event could be the trigger action generated by the other DIC.

In any case, trigger actions computed by the TPU can be used for (but not limited to) the following uses: (i) determine the beginning and/or the end of the sampling period of one or more sampled signals for design visibility; (ii) initiate the overwrite of one or more patch registers for design patching; (iii) provide a sampling clock in case none of the system clock signals shall be used; (iv) notify the communication controller within the DIC that one or more trigger events have occurred, thereby notifying the HDL-based hardware debugger; (v) communicate trigger events outside the electronic system to attached devices through externally connected signals; (vi) communicate with sub-systems inside the electronic system (e.g., during hardware/software co-debugging of embedded CPUs, trigger actions may be used as notification signals going into interrupt inputs of the CPUs); and (vii) connecting with the trigger event inputs of another DIC (e.g., when multiple DICs are daisy-chained to support debugging of multi-chip systems).

A trigger action can also be used to trigger multiple components. A trigger action group is a unique combination which comprises one or more units of design visibility and/or design patching circuitry which is/are controlled by the same trigger action. The internal structure of the TPU can be (but is not limited to) the following: (i) A simple TPU can be used where each trigger event issues exactly one and only one trigger action. (ii) A TPU can include a configurable combinational network where all the trigger events are inputs to the combinational network and trigger actions are outputs of the combinational network. For example, the configurability can be provided by a Random Access Memory (RAM) which can be configured by the HDL-based hardware debugger and act as look-up tables to implement a wide range of different boolean combinational functions. (iii) A TPU can be a configurable finite state machine where trigger events are inputs to the state machine and trigger actions are outputs of the state machine. In one example, the configurability is provided by a set of registers or a Random Access Memory (RAM) which defines the behavior of the finite state machine and which can be configured by the HDL-based hardware debugger. (iv) A TPU can be a pipelined CPU. The trigger events to be processed can flow into the TPU as input data, the trigger actions to be issued can be represented as output data of the CPU, the instruction code of the CPU can implement complex relationships between the trigger events which produce the trigger actions. The trigger action computations may not be finished until a number of clock cycles after the the trigger events flow into the TPU. Consequently, the design visibility circuitry should have enough memory to store the data which corresponds to the latent trigger actions. Also, the HDL-based hardware debugger should understand the latency of the trigger actions to correctly associate non-latent sampling data to the latent trigger actions.

Although the instrumentation techniques discussed above pertain to digitial signals, it should be understood that these same techniques can also apply to the digital portion of mixed-signal designs. Still further, with respect to the analog portion of mixed signal designs, an, analog signal can be made visible and also can be used to form trigger conditions. In one embodiment, the analog signal can be made visible by connecting it to an analog-to-digital converter (ADC) which has been added to the DIC. The digital outputs of the analog-to-digital converter can then be monitored using the design visibility techniques previously mentioned. A user interface can convert the digital data back to an analog representation for display to the designer. The analog signal can be used to form a trigger condition by expressing the trigger condition in terms of the digital outputs of the analog-to-digital converter. Additionally, a graphical user interface (e.g., the graphical user interface **704** of FIG. **7**A) can convert an analog trigger threshold set by the electronic system designer to an appropriate set of digital values which can be used to configure the trigger condition.

As noted above, the DIC can be provided within the DUT in either a centralized or distributed manner. More particularly, in order to minimize the impact of the DIC on the electronic system hardware, the DIC can be structured as a monolithic block or as distributed circuitry. The option to choose between these two structures allows the trade-off of area, delay, power consumption, routability, and/or testability of the hardware required for the DIC. As a monolithic block, all signals to be monitored for trigger detection or to be sampled and/or patched are physically routed from their source to the DIC region where the trigger condition detection and/or the signal value sampling/patching is physically placed. As a distributed DIC, the circuitry comprising the DIC is placed close to the signals used for triggering, sampling, and/or patching. For a monolithic DIC block, resource sharing to reduce the area and power consumption overhead becomes an option. These gains are offset by the increased delay and area needed for the long routes to the DIC block. A distributed DIC, however, will not offer any resource sharing, but promises short routes and therefore less impact on the delays and the routability.

Moreover, the monolithic or the distributed structure for the trigger detection circuitry can be selected independently from the monolithic or the distributed structure for the signal value sampling, patching, and storing circuitry. A special case of DIC structure is a DIC with monolithic trigger detection circuitry and monolithic signal value sampling and/or patching circuitry. The trigger detection and signal value sampling and/or patching circuitry share the same signals. In such a structure, trigger conditions can only be expressed using signals which are also sampled.

FIG. **18** is a flow diagram of HDL-based hardware debugging processing **1800** according to one embodiment of the invention. The hardware debugging processing **1800** is performed after the electronic system has been fabricated to include a customized DIC.

The hardware debugging processing **1800** initially starts when the HDL-based hardware debugger is initiated **1802** on a host computer. The HDL-based hardware debugger is preferably a software program that operates on the host computer. Next, the host computer couples **1804** with the operating fabricated electronic system. For example, this coupling **1804** can occur through cables that couple the host computer to the communication controller **816** of the DIC **800**. The DIC **800** can be considered part of the DUT or part of the electronic system. Thereafter, when debugging is to be performed, the DIC is configured **1806** for examination and/or modification of the fabricated electronic system. Here, for example, the configuration registers **814** of the DIC **800** can be configured **1806** to perform the appropriate examination and/or modification of the fabricated electronic system (namely, the DUT therein). Next, the fabricated

electronic system is operated **1808** in the target environment and at speed. In other words, the fabricated electronic system is the actual hardware that is produced and then operated in its normal operating environment (target environment) and at its normal speed of operation. Hence, this facilitates debugging of the hardware (e.g., fabricated electronic system) in its actual environment and at its actual speed. Thereafter, HDL-based hardware debugging is performed **1810** on the operating fabricated electronic system. The HDL-based hardware debugging thus interacts with the user to reference lines or areas of the HDL description associated with the electronic system. As a result, users are able to analyze, diagnose, and debug functional failures of the electronic system at the HDL level, and users are able to interact with the electronic system at the HDL level to set trigger conditions and examine and/or modify the electronic systems behavior. Following the operation **1810**, the hardware debugging processing **1800** is complete and ends.

Once the electronic system **104** having the DUT **102** with the incorporated DIC **106** has been fabricated, the HDL-based hardware debugger **122** can operate to debug the DUT **102**. The HDL-based hardware debugger **122** interacts with a user through one or more user interfaces and interacts with the DIC **106** through a host communication controller. The HDL-based hardware debugger. **122** can, for example, operate to support one or more of the following functions: (1) browsing the original HDL description for the HDL design; (2) activating particular trigger conditions out of the set of possible trigger conditions implemented in the DIC; (3) de-activating particular trigger conditions out of the set of activated trigger conditions; (4) temporarily disabling trigger conditions out of the set of previously activated trigger conditions; (5) enabling temporarily disabled trigger conditions; (6) activating signals to be sampled out of the set of possible signals in accordance with the implementation of the DIC; (7) de-activating signals out of the set of signals which were activated for sampling; (8) temporarily disabling signals out of the set of signals activated for sampling; (9) enabling temporarily disabled sampling signals; (10) activating signals to be patched out of the set of possible signals in accordance with the implementation of the DIC; (11) de-activating signals out of the set of to-be-patched signals; (12) temporarily disabling signals out of the set of signals activated for patching; (13) enabling temporarily disabled patching signals; (14) translating HDL-based trigger conditions given by the designer to the proper register configuration of the DIC; (15) associating trigger conditions with the clock/begin/end events of sampling and/or patching circuitry; (16) controlling execution of the DIC at run-time such as starting, stopping, single-stepping, running for a given number of cycles, resetting, etc.; (17) capturing the entire or the partial state of the HDL design, downloading it off the DIC, and storing it in the proper databases; (18) translating the DIC status registers and the sampled signal values back to the HDL source code; (19) displaying the DIC status in one or more formats, including the current data as well as data history; (20) displaying the signal sampling data in one or more formats, including the current data as well as data history; (21) interfacing with other debugging tools, such as functional simulators and software debuggers; (22) performing license checks to determine the legality of running the DIC; and (23) performing version checks of the DIC, and consistency checks of the DIC and the design instrumentation database.

FIG. **19** is a data flow diagram of a debugging process **1900** performed by a HDL-based hardware debugger according to one embodiment of the invention. An activation user interface **1902** displays the original HDL description **304** and provides the designer with a method to activate and de-activate break-points and other trigger conditions and to activate and de-activate signals for sampling and/or patching. Once signals for sampling and/or patching are activated, the activations may be grouped together to form a unique trigger action group. Each trigger action group then gets one or more trigger condition associated therewith that control the trigger action group. These activations are used by the HDL-based hardware debugger to configure the DIC at run-time.

Additional details on trigger condition activations are as follows. The structure of the DIC limits trigger conditions to the set of locations (for break-points) and explicit trigger condition expressions (for watch-points) in the HDL description **304** which were selected or implied during design instrumentation. Additional hardware restrictions of the DIC may also limit the activation of trigger conditions in certain cases. In accordance with the structure of the DIC, an active break-point database **1904** lists the status type of each trigger condition implemented in the DIC as one of: possible (i.e., the corresponding trigger condition can be activated); activated (i.e., designer has activated); and forbidden (i.e., the trigger condition cannot be activated due to a mutual exclusivity relationship with one or more currently activated trigger conditions. Initially, a break-point manager **1906** copies over the set of trigger conditions from the break-point database **602** into the active break-point database **1904** and marks all entries as possible. To guide the designer in his activations, the user interface **1902** reads the active break-point database **1904** and displays the current status for each trigger condition listed. Whenever the designer activates a trigger condition out of the set of possible trigger conditions, the user interface **1902** marks the trigger condition as activated in the active break-point database **1904** and notifies the break-point manager **1906**. Likewise, whenever the designer de-activates a trigger condition out of the set of activated trigger conditions, the user interface **1902** marks the trigger condition as de-activated in the active break-point database **1904** and notifies the break-point manager **1906**. The break-point manager **1906** applies the rules in the break-point database **602** which describe the interdependencies of all trigger conditions and their mutual exclusivity to the current setting in the active break-point database **1904**. Under such rules, any trigger condition which is mutually exclusive with the activated (or de-activated) trigger condition is marked as forbidden (or possible), as appropriate.

Additional details on signal sampling and patching activation are as follows. To utilize the signal sampling and patching circuitry in the DIC, the designer activates signals for sampling and/or patching, groups these activations into one or more trigger action groups, and associates one or more trigger conditions by which each trigger action group is controlled. For patching, the designer also specifies one or more patch values and the trigger condition settings under which each patch value shall be applied. To reflect limitations of the DIC in the sharing of sampling and/or patching resources, a similar activation mechanism for signal values exists as for trigger conditions. An active signal value database **1908** lists the status type of each signal that has been made visible as one of: possible (i.e., the signal can be activated for sampling and/or patching); activated (designer has activated); and forbidden (i.e., the signal cannot be sampled/patched due to a mutual exclusivity relationship with one or more currently sampled/patched signals). Initially, a signal value manager **1910** copies over the set of all signals listed in the signal value database **604** into the active

signal value database **1908** and marks them as possible. To guide the designer in making activations, the user interface **1902** reads the active signal value database **1908** and displays the current status for each signal listed. Whenever the designer activates a signal out of the set of possible signals, the user interface **1902** marks the signal as activated in the active signal value database **1908** and notifies the signal value manager **1910**. Likewise, whenever the designer de-activates a signal out of the set of possible signals, the user interface **1902** marks the signal as de-activated in the active signal value database **1908** and notifies the signal value manager **1910**. The signal value manager **1910** applies the rules in the signal value database **604** which describe the interdependencies of all signals and their mutual exclusivity to the current setting in the active signal value database **1908**. Under these rules, any signal which is mutually exclusive with the activated or de-activated signal is marked as forbidden or possible, as appropriate.

After the various activations have been made with respect to run-time configuration of the DIC, the designer notifies a run-time controller **1912** through a run-time user interface **1914** to configure the. DIC. Using the rules in the DIC database **736**, a DIC configuration manager **1916** translates the information in the active break-point database **1904** and the active signal value database **1908** to the proper values for the DIC's configuration registers and writes a DIC configuration file to a DIC configuration database **1918**. A register-to-physical address translator **1920** (R2P translator) then accesses the R2P database **614** (i.e., register-to-physical address translation table) and translates the DIC configuration file to the proper physical memory locations within the DIC and produces a raw configuration file **1922**. The raw configuration file **1922** is then uploaded into the DIC by a host communication controller **1924** that communicates with the client communication controller **816** inside the DIC **800**. This configures the DIC to detect the proper trigger conditions and to sample/patch the proper signals as specified by the designer. For efficiency, the host communication controller **1924** provides a method of handling incrementally the raw configuration file **1922** and uploads only changed data into the DIC **800**. The host communication controller **1924** communicates with the client communication controller **816** by transmitting control signals, uploading data, receiving control signals, and downloading data via one or more connections (communication links). When at least one trigger condition is detected, the trigger processing unit **808** inside the DIC **800** informs the run-time controller **1912** via a communication link connected to the host communication controller **1924**.

The HDL-based hardware debugger also performs signal value examination. When the HDL-based hardware debugger has been notified that one or more trigger conditions have been detected, the host communication controller **1924** downloads data from the DIC and stores it in a raw status file **1926**. This raw status data is then split by the R2P translator **1920** into data from the DIC status registers and data from the signal value sample memory. The data from the DIC status registers is stored in a DIC status database **1928**. The DIC configuration manager **1916** accesses the DIC database **736** and the active break-point database **1904** and determines which of the activated trigger conditions were actually detected. The detected trigger conditions are then marked as triggered in the active break-point database **1904**. The activation user interface **1902** thereafter displays the detected trigger conditions as marked. On the other hand, the data (values) of the sampled signals from the signal value sample memory are stored in a system state database **1930**.

A history manager **1932** picks up values of the sampled signals from the system state database **1930**, analyzes the history based on the sample clock periods, and appends them to a signal value history database **1934**. The signal value history database **1934** provides a method of storing sampled signals for particular sample times. A signal value resolver **1936** reads the signal value history database **1934**, resolves the data back to HDL identifiers by applying the resolution rules of the cross-reference database **612**, and writes the data into a global signal value database **1938**. Any re-organization and/or transformation of the signal data to support HDL identifiers with complex values (for example multi-bit or symbolically encoded values) can also be performed by the signal value resolver **1936**. Signals, whether selected or implied, which have not been directly sampled but which can be derived from sampled values, are calculated by the signal value resolver **1936** and stored in the global signal value database **1938**. The global signal value database **1938** comprises the current value and the value history of all the signals, sampled and/or derived. The value history can be used for display to the designer or for further processing. A format translator **1942** accesses the global signal value database **1938** and translates the data into one or more different file formats. For example, the format translator **1942** can produce vector change dump files **1944**, wave vector files **1946**, or debug data files **1948** suitable for further processing by third party tools such as simulators. The display manager **1940** gets directions from a display user interface **1950** about which values to query for display from the global signal value database **1938**. The display user interface **1950** uses the original HDL **304** to provide a method for HDL-based signal examination for the designer.

When software debugging is also to be performed, the debugging process **1900** can include a software debugger interface **1960** and a software debugger **1962**. Additional details on software debugging are provided below with respect to FIG. **20**.

Still further, the HDL-based hardware debugger can perform check-point processing. The system state of the HDL design including the DIC is represented by the values of the electronic system's registers and inputs. The HDL-based hardware debugger provides a method for saving and restoring the system state to the system state database **1930**. Depending on whether all the registers and inputs are sampled, or only some of them, the system state can be saved in full or partially. Sometimes a partial system state is sufficient, sometimes the full system state is necessary. The capability to save and restore the electronic system's state can be used for many applications. As examples, one application can set the electronic system to a known state during HDL-based hardware debugging, and another application can integrate the present invention with functional simulators.

HDL-based hardware debugging using the sampling and trigger detection methods described in the present invention still may not give every detail of every internal signal like an event-driven functional simulator may give. Thus, it may be desirable to combine both approaches and have one system, where the HDL-based hardware debugging techniques are used when there is a need for a high execution speed and/or real-time behavior, and where a functional simulator is used for time periods which are not speed-critical but where a great level of detail is needed. In order to combine both styles, the HDL-based hardware debugger described in FIG. **19** provides a way to exchange information about the system state with a functional simulator. Most functional simulators provide a method for saving the simulation state of a simulation model of the HDL design in a checkpoint file using a variety of different file formats. The file formats can be processed by a checkpoint manager **1952**. For uploading the state of the simulation model into the HDL-based hardware debugger, a simulator checkpoint input file **1954** is translated by the checkpoint manager **1952** using the cross-reference database **612** and stored in the system state database **1930**. To start the functional simulation from a given state of the HDL design, the checkpoint manager **1952**, using the cross-reference database **612**, can convert the contents of the system state database **1930** into a simulator checkpoint output file **1956** in a format suitable for a functional simulator. A checkpoint file **1958** can be used for storing and retrieving the system state of the DUT, for example, for subsequent runs of the HDL-based hardware debugger.

Still further; the HDL-based hardware debugger can perform mismatch processing. The mismatches can occur between different runs of the DUT. In some situations it may be useful to find mismatches in the sampling data gained from running the same version of the DUT under different conditions. For example, this could be used for verifying that the functionality of an HDL design has not changed after the HDL design has been modified. In some other situations it may be useful to find mismatches in the sampling data gained from running two different versions of the same DUT under identical conditions. To make it easier for the designer to understand any mismatches found, the HDL-based hardware debugger can relate mismatches back to the original HDL description and display both sets of signal values. The mismatches can also occur between the HDL description and the DUT. In some situations it may be useful to compare the functional behavior of a fabricated electronic system with the functional behavior of the HDL description of the electronic system. A mismatch in the comparison means that some step in the design flow was incorrect. The electronic system need not be fully instrumented since some functional mismatches can be caught with partial instrumentation.

A representative method for performing such a comparision is as follows: First, the HDL design is instrumented. The instrumentation is most useful when the design visibility covers the entire system state. Second, with the instrumentation enabled, run the DUT in an environment and at a speed for which it was targeted. Third, store all sample data gained from the operation of the DUT. Fourth, starting with the earliest clock cycle for which sample data is available, format the sample data so that it will be accepted by a functional 'simulator. Fifth, use the formatted data to set the initial state of the HDL design in a functional simulation of the HDL design. If the HDL design was partially instrumented, substitute the appropriate "UNKNOWN" simulation value for any un-instrumented inputs or storage elements in the circuit. Sixth, use the functional simulator to calculate the values of the storage elements in the next clock cycle given the initial state set above. Seventh, compare the calculated values of the storage elements with the sample data for the next clock cycle and note any mismatches. If the HDL Design was partially instrumented, any comparisons to an "UNKNOWN" value are NOT a mismatch. Eighth, take the sample data for the inputs and storage elements from the next cycle, format as appropriate, and use such to re-set the initial state of the functional simulator. Ninth, while there is more design visibility data left, return to the sixth operation. The mismatches found in the seventh operation are potential problems and should be investigated by the designer. To make it easier for the designer to understand any mismatches found, the HDL-based hardware debugger can relate the mismatches back to the original HDL description and display both sets of signal values.

In the above representative method, mismatches are found by comparing the sampling data with the values calculated from the HDL description by a functional simulator. Obviously, the full power and generality of a functional simulator is not required here. Any method that can calculate delay-independent functional values from an HDL description can be used to find mismatches. For example, the cross-reference database can contain a representation of the necessary function of the HDL description and can be used to calculate the values directly.

FIG. **20** is a block diagram of a hardware/software co-debugging system **2000** according to one embodiment of the invention. The hardware/software co-debugging system **2000** is generally similar to the hardware debugging system **100** of FIG. **1**A or the hardware debugging system **150** of FIG. **1**B, but the DUT **102** includes not only the DIC **106** but also a Central Processing Unit (CPU) **2002**. The hardware/software co-debugging system **2000** thus permits debugging not only software that runs on the CPU **2002** but also debugging the DUT **102**. For debugging the software that runs on the CPU **2002**, a software debugger **2004** is used. The software debugger **2004** is a software program that runs on a host computer and controls and observes the execution of the computer software code which runs on the embedded CPU **2002**. For example, the software debugger **2004** can be the software debugger **1962** illustrated in FIG. **19**. The software debugger **2004** allows program break-points to be set. Those program break-points define the condition upon which the program execution is halted such that the designer can examine the operation of the software program. If the embedded system (CPU **2002**) cannot be halted, the software debugger **2004** takes a snapshot of the software program's state for examination instead.

FIG. **21** is a block diagram of a hardware/software co-debugging system **2100** according to one embodiment of the invention. The hardware/software co-debugging system **2100** is generally similar to the hardware/software co-debugging system **2000** of FIG. **20** with the addition of an In-Circuit Emulator (ICE) **2102**. The ICE **2102** interfaces the software debugger **2004** with the CPU **2002**. The ICE **2102** is, more generally, a debugger interface. An example of such a debugger interface is described in "*The Nexus* 5001 *Forum Standardfor a Global Embedded Processor Debug Interface*," which is available by the IEEE-ISTO in Piscataway, N.J., and which is hereby incorporated by reference. It should also be noted that as shown in FIG. **21** the CPU **2002** may not be part of the DUT **102**. In general, the software being debugged can execute on the CPU **2002**. The CPU **2002** need not be within the DUT **102**. In other words, the CPU **2002** can be part of the electronic system **104** or can even be external to the electronic system **104** if coupled thereto.

Concurrent debugging of the HDL design and the CPU software deals with the following two cases: (i) a trigger condition set in the HDL-based hardware debugger and detected at run-time in the DIC; and (ii) a program break-point is set in the software debugger **2004** and detected in the CPU **2002** and/or the ICE **2102**.

The setting and detecting of at least one trigger condition in the DIC and examining the operation of the HDL design and/or the software program can be done in the following operations. First, a trigger condition is set in the HDL-based hardware debugger (HHD) **122**. Second, the HHD **122** configures the DIC **106** via a communication link **2104**.

Third, if the trigger condition is met, one or more trigger actions are issued in the DIC **106**. One trigger action in the DIC **106** notifies the HHD **122** via the communication link **2104**. One trigger action in the DIC **106** notifies the CPU **2002** via a communication link **2106**. On the CPU side, the communication link **2106** may be connected to an interrupt input. Fourth, the HHD **122** then downloads the DIC status and the sample values for processing and display. Fifth, the CPU **2002** then notifies the ICE **2102** via a communication link **2108**. Sixth, the ICE **2102** then notifies the software debugger **2004** via the communication link **2110** that a trigger condition was detected. Alternatively, the HHD **122** can directly notify the software debugger **2004** via the software debugger interface **1960**. Seventh, the software debugger **2004** then takes a snapshot of the current status of the software program and/or halts the program's execution. Eighth, the status and the history of the operation of the HDL design and the software program can then be examined in the user interface **2116**.

The setting and detecting of at least one trigger condition in the software debugger **2004** and examining the operation of the HDL design and/or the software program can be done in the following operations. First, a program break-point is set in the software debugger **2004**. Second, the software debugger **2004** sets up the ICE **2102** via the communication link **2110**. The ICE **2102** monitors some internal portions of the CPU **2002** (for example the instruction pointer counter) to determine whether the program break-point is reached. Third, if the program break-point is reached, the following actions are issued: (i) one action issued by the ICE **2102** notifies the software debugger **2004** via the communication link **2110**; and (ii) another action issued by the CPU **2002** notifies the DIC **106** via the communication link **2106**. On the DIC's side the communication link **2106** can be connected to an external trigger event input. Fourth, the software debugger **2004** then takes a snapshot of the current status of the software program and/or halts the program's execution. Fifth, the DIC **106** then processes the trigger event(s) and informs the HHD **122** via the communication link **2104**. Sixth, the HHD **122** then downloads the DIC status and the sample values for processing and display. Seventh, the status and the history of the operation of the HDL design and the software program can then be examined in the user interface **2116**. Depending on the debugging tools utilized, the user interface **2116** can be either integrated into the HHD **122** and/or into the software debugger **2004**.

The hardware debugging system according to the invention can have numerous features. The hardware-debugging system can, for example, be the hardware debugging system **100** illustrated in FIG. **1**A or the hardware debugging system **150** illustrated in FIG. **1**B. Exemplary features of the hardware debugging system might include one or more of those features examined below.

One exemplary feature pertains to HDL-based hardware debugging. While debugging an electronic system, the values of numerous signals may be examined. Relating these values of numerous signals back to the HDL description of the electronic system allows a user (e.g., designer) to gain an understanding of the operation of the electronic system. This enables the debugging to be performed at the same level of abstraction and using the same text description that the designer of the electronic system used to design and implement the electronic system. During the design phase of an electronic system, there are many transformations made to the HDL description to produce the fabricated electronic system. While such transformations conventionally often make it very difficult to difficult to relate a signal in the fabricated electronic system to the HDL description, the invention is able to relate the signals automatically and thus provides an efficient and effective approach to debugging the electronic system.

Another exemplary feature pertains to the ability to debug in a target environment at target speed. Performing HDL-based hardware debugging, while the electronic system is running in an environment and at a speed for which the HDL design is targeted, provides the following benefits: high processing bandwidth, real-time debugging, and no need for testbenches. During debugging, all operations may take the same time as in normal (non-debugging) operation which provides high processing bandwidth. For example, booting an operating system is a task which requires many clock cycles and is usually too time consuming to be done in functional simulation. In HDL-based hardware debugging, booting may take the same amount of time which it takes in normal (non-debugging) operation of the electronic system. Consequently, the designer can re-run the booting as often as necessary to fully debug the electronic system. Real-time debugging is useful for debugging electronic systems which have to maintain a specified real-time behavior in the sense that certain operations must be performed within a very well-defined time limit. Further, since a failure within the electronic system can be observed, analyzed and diagnosed within the target environment, there is no need to reproduce the failure in a model of the target environment, such as a testbench, for functional simulation or emulation.

Another exemplary feature pertains to the ability to communicate with hardware not instrumented. In some cases it may be important for a DIC to communicate with other hardware that was not, or could not be, instrumented. Such communication can be done via dedicated ports of the DIC which can be connected to other devices in the electronic system, or to portions within the same device the DIC resides in. These ports can be uni-directional or bi-directional. One example use of such ports is to communicate one or more trigger actions to another part of the electronic system. Another example is to connect an interrupt signal from another device to the DIC. The interrupt signal can then be used as a trigger event inside the DIC.

Still another exemplary feature pertains to the ability of the HDL-based hardware debugger to communicate with other systems. The HDL-based hardware debugger is a software system which can communicate with other software or hardware systems. The communications can allow transfer of information into, or out of, the HDL-based hardware debugger. For example, an electronic system may be able to execute a software program and in such case the HDL-based hardware debugger can communicate with a software tool which can debug the software program. The HDL-based hardware debugger may also communicate with hardware devices. For example, the HDL-based hardware debugger may send reset signals to hardware devices which connect to the DUT being debugged. In one embodiment, the connection to other hardware devices is used to form a JTAG daisy-chain.

Yet another exemplary feature pertains to the ability to provide hardware and/or software debugging. Some electronic systems have the capability to execute a software program. Software tools exist to debug the programmable hardware. It is advantageous for the designer of the electronic system to have the capability to debug both the hardware and software aspects of the electronic system concurrently. The HDL-based hardware debugger can enable such a capability by debugging the hardware of the electronic system and interfacing with software debugging tools. Interfacing with the software debugging tools can be done by using communication methods previously described. The combined hardware and software debugging system allows the designer to concurrently debug an entire electronic system including both hardware and software aspects.

The HDL-based hardware debugging can be used in many different applications. Different embodiments or implementations of the invention may be used in one or more of the following applications. Several example applications for the HDL-based hardware debugging are examined below.

One exemplary application for the HDL-based hardware debugging is property checking at target speed. Functional simulation alone cannot guarantee that a HDL design meets a functional specification for the HDL design. Consequently, additional methods of gaining confidence in the correctness of the functionality of a HDL design are necessary. A designer can increase the level of confidence in the function of the HDL design by adding DIC which can detect when the HDL design is operating contrary to its functional specification. The DIC. can provide property checks to assist the designer with identifying various conditions. The designer might also build in property checks to handle anticipated difficulties. Typically, during HDL-based hardware debugging, the property checks are activated and the electronic system is allowed to run in an environment and at a speed for which it is targeted. If the electronic system operates in a manner that causes a property check to issue a trigger event, the designer has found a potential problem.

Software tools exist that formally prove that certain property checks will never be triggered under any operating conditions of the design. Unfortunately, such tools may have tremendously long run-times since they must exhaustively analyze the design. The HDL-based hardware debugging approach does not have the problem of long run-times since all property checking is done in hardware that is running at target speed.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware debugging of errors in functional specifications. Some of the hardest functional failures to diagnose are misunderstandings of the target environment the electronic system is designed to work in. Such misunderstandings may lead to mistakes in the functional specification of the electronic system. Hence, comparing the implementation of the electronic system with its specification will not reveal such functional failure. However, the functional failure will become apparent when the electronic system is run in its target environment. While conventional methods for debugging, such as logic analyzers, can connect to accessible pins to monitor the operation of the electronic system within its target environment, these conventional methods do so only at a very low level of abstraction. In contrast, the HDL-based hardware debugging system according to the invention supports analysis, diagnosis and debugging of functional failures due to mistakes in the functional specification. First, there is no need to reproduce the problem in a testbench because the hardware itself is tested in its target environment. The ability to observe the HDL design while it is running in its target environment at the targeted speed allows the designer to immediately gather information about the electronic system as well as the environment the system is running in. Second, the information gathered is related back to the HDL description, which is the highest level of abstraction.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware debugging of design errors. Design errors stem from mismatches in the behavior of the HDL description written by the designer and the functional specification. Conventionally, such problems are normally debugged by reproducing the observed error in a testbench for a functional simulator. Though functional simulation gives information at a very detailed level, creating and enhancing a testbench to reproduce a functional failure is often a very tedious and difficult task. In contrast, with HDL-based hardware debugging provided by the invention, there is no need to reproduce the problem in a simulation model. By running the electronic system in the environment where the design error becomes apparent, sampling the desired portions of the system state, and analyzing the observed behavior which is related back to HDL identifiers, a functional failure can quickly be diagnosed. Having gained an understanding of the operation of the system, the designer then can use patching to apply a fix. Then, by re-running the patched HDL design in the target environment, the designer can check whether the problem is fixed. In addition, the HDL-based hardware debugger can write out the sampled information in a format suitable for a functional simulator tool (check-pointing) so that the designer can use their preferred analysis tools. The above-described check-pointing mechanism to forward the sampled information to functional simulation can additionally be used.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware debugging of tool errors. Tool errors are functional failures which happen when, for example, a synthesis tool involved in HDL design process does not transform the HDL description into a correct fabricated design. Such errors manifest themselves as mismatches between the functional specification and the functionality of the fabricated design, therefore debug techniques which work on the HDL description cannot be used to debug such errors. However, since HDL-based hardware debugging works on the instrumented design which was produced by the erroneous tool, the symptoms are able to be displayed to the designer for diagnosis.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware timing error analysis. Examples of timing errors in an HDL design are race conditions as well as setup and hold time violations in the hardware implementation. One symptom of a timing error is that some registers do not store the correct, expected values. This symptom is easily detected using the method of checking for mismatches between the functional simulation result and the values sampled by the DIC. When the designer examines the values of the circuitry that drive the erroneous register, the cause for the symptom can be quickly diagnosed. The impact of signal noise on the behavior of the electronic system can also be similarly analyzed and diagnosed.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware fault analysis. Faults stem from manufacturing defects. When faults show up occasionally in a non-reproducible manner for one particular device or for only certain devices out of a batch of other devices, diagnosis becomes very difficult. The HDL-based hardware debugging can be used to diagnose faults, and relate them back to HDL identifiers to provide leads for the fault analysis. Detection of faults is identical to the detection of timing errors and is done by checking for mismatches between functional simulation results and values sampled by the DIC. The ability to relate sample values to the HDL description is a significant advantage since the designer can quickly identify the problem. Once the problem is located in the HDL description, the designer can trace the

problem all the way to the layout level to determine the physical location of the defect or defects that caused the fault. The designer can then perform very precise design rule checks. The ability to limit the area for the design rule checks to the neighborhood of the defect location greatly reduces the effort. If the fault is caused by a design rule violation, it thus can be quickly found and fixed. Knowing the context of the fault may also help to improve the manufacturing test program and/or improve the manufacturing yield.

Another exemplary application for the HDL-based hardware debugging is HDL-based critical-path analysis of hardware. To analyze the timing and identify critical paths in the HDL design, the following is one method that can be used. Initially, the HDL design is run at the target speed in the target environment and using some predetermined trigger conditions, some predetermined signals are sampled and the value history is stored. Then, iteratively, the frequency of one or more clock signals is step-wise increased, the HDL design is run at the increased clock speed/speeds while the HDL-based hardware debugger samples the very same signals under the very same trigger conditions as performed in the initial operation. For each iteration, the HDL-based hardware debugger checks for a mismatch between the current sampling values and the initial sampling values. If a mismatch is detected, the HDL-based hardware debugger informs the designer about the mismatch and the designer can then analyze the portion of the HDL design in which the mismatch occurred. The portion of the HDL design in which the mismatch occurred is likely to be a part of the critical path of the electronic system.

Another exemplary application for the HDL-based hardware debugging is analysis, diagnosis and debugging-of environmental errors. Environmental factors such as temperature, pressure, radiation, electromagnetic fields, and aging effects may cause transient or permanent failures of the electronic system. Sometimes an electronic system works reliably in the field for years until aging and/or environmental factors cause functional failures. If parts of the electronic system have been instrumented, the invention can be used to diagnose the problem quickly by looking for mismatches between the function of the electronic system and sampled data taken from the fabricated design. If the electronic system has been instrumented with design patching, the electronic system might be patched to restore the proper behavior.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware power analysis. Power analysis of the electronic system needs to know about the realistic stimuli and transitions in the electronic system to come up with an accurate estimation of the power consumption. In a hardware power analysis application according to the invention, the system state of the HDL design running in the target environment at target speed is sampled and stored by the HDL-based hardware debugger and transformed into the proper format for describing such stimuli and transitions which can be processed by tools which are specialized for power calculations.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware regression testing. For regression testing of changes to the hardware design, the invention can be used as follows. An initial version of the instrumented HDL design, which itself has been tested and found correct, is run with some predetermined trigger conditions and some predetermined signals to be sampled. The sample values and their history are stored as a "golden" reference file. Each HDL design which includes a design change is then run again using the same trigger conditions and sampling the same signals at the same events. The HDL-based hardware debugger then checks for mismatches between the reference file and the current sampling data and issues warnings if mismatches are detected. Accordingly, the design change that introduced the mismatched behavior can be quickly isolated and fixed.

Another exemplary application for the HDL-based hardware debugging is HDL-based testbench optimization. The reference file of the hardware regression testing application can be used as stimuli to create a new testbench for functional simulation, or optimize an existing testbench to more closely mimic the behavior of the target environment.

Another exemplary application for the HDL-based hardware debugging is HDL-based hardware device driver debugging. The debugging of a particular device driver which interacts with the HDL design is similar to hardware/software co-debugging. The designer is thus able to see the effects of the device driver on the HDL design it interacts with immediately. In numerous applications of the invention, an electronic system shall be debugged after it has initially executed certain setup operations. Having the electronic system execute the operations for setup can be slow, tedious, and cumbersome. For example, an operating system may be booted and many other device drivers may be loaded before a particular device driver and the hardware used by it can be debugged. Now, if the designer has to iterate over the initialization many times, it is advantageous that the system state right after the initialization be saved and restored before each iteration (e.g., system state database **1930** of FIG. **19**). The restoring will operate to bring the HDL design into exactly the same post-initialization state.

Another exemplary application for the HDL-based hardware debugging is HDL-based software quality analysis in target hardware. The invention can also be used in regression testing and software quality assurance of the software that runs on the HDL design. If one or more software regression tests fail, the HDL-based hardware debugger can be used to quickly diagnose the failure.

Another exemplary application for the HDL-based hardware debugging is HDL-based embedded systems debugging. Software that runs on an embedded CPU within the HDL design is able to be debugged by a software debugger. The software debugger can communicate with a HDL-based hardware debugger that debugs the hardware of the HDL design.

Still another exemplary application for the HDL-based hardware debugging is in-field support. A common use of the HDL-based hardware debugging system is to instrument an electronic system and then use the HHD **122** to debug the system. After debugging and fabrication, copies of the fabricated electronic system can be distributed to the designer's customers. At this point, the DIC **106** can be used in an in-field mode. In the in-field mode, the DIC **106** is used to diagnose failures that occur while the electronic system is being used by customers. The DIC **106** still resides in the fabricated electronic system but the DIC's normal state is disabled. It will be enabled if there is a problem with the electronic system. In addition, a specially trained service personnel can be sent to the customer's site. The personnel can attach the instrumented electronic system to a portable host computer which runs the HHD **122**, activate the DIC **106**, and debug the HDL design in the customer's environment. If the instrumented electronic system has been designed with a telecommunications link between the DIC **106** and the HHD **122**, remote debugging may avoid the need for service personnel to be sent to the customer's site.

Yet another exemplary application for the HDL-based hardware debugging is hardware performance monitoring. Often it is important for a hardware system designer to monitor the performance of a hardware system in order to understand and optimize the system. This can be done by a software simulation of the system. Unfortunately, this has the drawback that it requires a model of both the electronic system and of the environment it operates in. By adding performance monitoring circuitry to the DIC **106** of the electronic system, the designer can monitor the performance of the fabricated electronic system operating in its target environment and at its target speed. The process of adding the monitoring circuitry begins with the instrumentor. The instrumentor displays the HDL description and enables the designer to add performance monitoring circuitry which relates to the HDL description. During debugging, the data from the performance monitoring circuitry is loaded from the DIC **106** to the HHD **122** after a specified number of clock cycles or in response to some trigger event. The HHD **122** then displays the data for the designer in the proper format. The circuit performance that can be monitored by this added circuitry is quite broad; for example, a circuit performance parameter in which there are events that can be counted—the number of times a First-In-First-Out (FIFO) queue overflows, a number of cache misses, etc. Further, average values, such as average stack depth, can also be monitored by using more complex circuitry.

Portions of the invention are preferably implemented in software. Such portions of the invention can also be embodied as computer readable code on a computer readable medium. The computer readable medium is any data storage device that can store data which can be thereafter be read by a computer system. Examples of the computer readable medium include read-only memory, random-access memory, CD-ROMs, magnetic tape, optical data storage devices, carrier waves. The computer readable medium can also be distributed over a network coupled computer systems so that the computer readable code is stored and executed in a distributed fashion.

The many features and advantages of the present invention are apparent from the written description and, thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

What is claimed is:

1. A machine-readable medium containing instructions that when executed on a data processing system causes the system to perform a method for debugging a fabricated integrated circuit containing an electronic circuit design, the method comprising:

receiving a high level HDL description of the electronic circuit design;

determining aspects of the electronic circuit design to be examined or modified during debugging;

determining additional circuitry to be incorporated into the electronic circuit design to facilitate debugging;

producing a modified high level HDL description of the electronic circuit design by incorporating an HDL description of the additional circuitry into the high level HDL description of the electronic circuit design;

storing information about the additional circuitry including relationships between signals of the electronic circuit design and portions of the modified high level HDL description; and

debugging the fabricated integrated circuit fabricated in accordance with the modified high level HDL description by interacting with the electronic circuit design using the additional circuitry and by operating to present debug information with respect to the modified high level HDL description or the high level HDL description.

2. The machine-readable medium as recited in claim **1**, wherein at least a portion of the high level HDL description is written in a language selected from the group consisting of VHDL, Verilog HDL, C, C++ and SystemC.

3. The machine-readable medium as recited in claim **1**, wherein debugging the fabricated integrated circuit does not require a testbench, wherein while debugging the fabricated integrated circuit, the fabricated integrated circuit is operating in its target environment without interruption and running at its target speed, and wherein the target environment includes real-time characteristics.

4. The machine-readable medium as recited in claim **1**, wherein the method further comprises:

relating the debug information back to the high level HDL description for the electronic circuit design;

enabling a user to determine the aspects of the electronic circuit design to be examined or modified during debugging through interactive selection;

customizing the additional circuitry for use with at least a portion of the electronic circuit design; and

altering the additional circuitry to trade-off debugging coverage versus area cost.

5. The machine-readable medium as recited in claim **1**, wherein the information about the additional circuitry further includes one or more trigger conditions and at least one of design control information, design visibility information and design patch information.

6. The machine-readable medium as recited in claim **1**, wherein the fabricated integrated circuit is part of an electronic system that also includes software, wherein the method further comprises debugging the software, wherein the fabricated integrated circuit includes a processor, wherein the software is executed by the processor, and wherein debugging the fabricated integrated circuit is synchronized with debugging the software.

7. The machine-readable medium as recited in claim **6**, wherein the hardware debugging system does not require a testbench, wherein while debugging the fabricated integrated circuit, the fabricated integrated circuit is operating in its target environment and running at its target speed, and wherein the target environment includes real-time characteristics.

8. The machine-readable medium as recited in claim **1**, wherein the HDL description contains a hierarchical structure of HDL building blocks, wherein the electronic circuit design includes both analog and digital aspects, and wherein the aspects of the electronic circuit design to be examined or modified during debugging are determined in different ones of the HDL building blocks of the hierarchical structure.

9. The machine-readable medium as recited in claim **1**, wherein the electronic circuit design includes at least one pre-designed block of circuitry having instrumentation circuitry, wherein the high level HDL description of the electronic circuit design is partitioned into two or more design parts, wherein the additional circuitry is partitioned into two or more partitions, each partition to debug a different part of the electronic circuit design, and wherein each design part contains its own additional circuitry to debug its corresponding design part.

10. A machine-readable medium containing instructions that when executed on a data processing system causes the system to perform a method for debugging an electronic system designed according to an electronic circuit design, the electronic circuit design being described by a high level HDL description, the method comprising:

receiving the high level HDL description of the electronic circuit design or a description derived therefrom;

determining aspects of the electronic circuit design to be examined or modified during debugging;

determining additional circuitry to be incorporated into the electronic circuit design to facilitate debugging;

incorporating the additional circuitry into the electronic circuit design;

storing information about the additional circuitry including relationships between signals of the electronic circuit design and portions of the high level HDL description; and

debugging the electronic system by interacting with the electronic circuit design using the additional circuitry and by operating to present debug information with respect to the high level HDL description.

11. The machine-readable medium as recited in claim **10**, wherein at least a portion of the high level HDL description is written in a language selected from the group consisting of VHDL, Verilog HDL, C, C++ and SystemC.

12. The machine-readable medium as recited in claim **10**, wherein the method further comprises:

identifying functional failures that result from one or more of design errors, tool errors and manufacturing faults; and

identifying functional failures that result from specification errors.

13. The machine-readable medium as recited in claim **10**, wherein debugging the electronic system does not require a testbench, wherein while debugging the electronic system, the electronic system is operating in its target environment and running at its target speed, and wherein the target environment includes real-time characteristics.

14. The machine-readable medium as recited in claim **10**, wherein the high level HDL description contains a hierarchical structure of HDL building blocks, wherein the electronic circuit design includes both analog and digital aspects, and wherein the method further comprises determining, in different ones of the HDL building blocks of the hierarchical structure, aspects of the electronic circuit design to be examined or modified during debugging.

15. The machine-readable medium as recited in claim **10**, wherein the method further comprises:

customizing the additional circuitry for use with at least a portion of the electronic circuit design;

altering the additional circuitry to trade-off debugging coverage versus area cost; and

implementing at least one of a design layout tool, a synthesis tool and a simulation tool.

16. The machine-readable medium as recited in claim **10**, wherein the electronic system includes hardware and software, wherein the method further comprises debugging the hardware and the software together, wherein the electronic system includes a processor, the processor to execute the software, and wherein debugging the electronic circuit is synchronized with debugging the software.

17. The machine-readable medium as recited in claim **10**, wherein the electronic system comprises at least one of an integrated circuit hardware product, a programmable integrated circuit and a printed circuit board with electronic components thereon, each of the integrated circuit hardware product, the programmable integrated circuit and the printed circuit board with electronic components thereon including at least a portion of the electronic circuit design.

18. The machine-readable medium as recited in claim **10**, wherein the electronic circuit design includes at least one pre-designed block of circuitry having internal circuitry, wherein the electronic circuit design is partitioned into different sections, each different section of the electronic circuit design containing its own portion of the additional circuitry for debugging its corresponding section of the electronic circuit design, and wherein the additional circuitry is partitioned into different sections, each different section of the additional circuitry to debug a different portion of the electronic circuit design.

19. A machine-readable medium containing instructions that when executed on a data processing system causes the system to perform a method for debugging an electronic system having instrumentation circuitry included therein, wherein the electronic system is described with a hardware description language (HDL), the method comprising:

activating at least one aspect of the instrumentation circuitry available for debugging the electronic system via the instrumentation circuitry, the aspect selected from the group consisting of design visibility, design patching and design control;

determining configuration information based on the certain design visibility, design patching or design control aspects that are activated;

configuring the instrumentation circuitry in accordance with the configuration information;

receiving debug data from the configured instrumentation circuitry operating within the electronic system;

translating the debug data into HDL-related debug information; and

relating the HDL-related debug information to the HDL description of the electronic system.

20. The machine-readable medium as recited in claim **19**, wherein at least a portion of the HDL description of the electronic system is written in a language selected from the group consisting of VHDL, Verilog HDL, C, C++ and SystemC.

21. The machine-readable medium as recited in claim **19**, wherein the HDL description is a high-level HDL description and the HDL-related debug information is described in a high-level HDL, and wherein the method further comprises displaying the HDL description with the HDL-related debug information related thereto.

22. The machine-readable medium as recited in claim **19**, wherein the method operates without any requirement for a testbench, wherein translating is performed automatically, and wherein the debug data includes at least status information or sampling data.

23. The machine-readable medium as recited in claim **19**, wherein activating operates to enable a user to activate the certain design visibility, design patching or design control aspects, and wherein activating is performed using a graphical user interface.

24. The machine-readable medium as recited in claim **19**, wherein the design control aspects include trigger conditions, and wherein activating operates to enable a user to set one or more trigger conditions from the trigger conditions available by the instrumentation circuitry.

25. The machine-readable medium as recited in claim **19**, wherein the electronic system includes a hardware portion and a software portion, and wherein the method further comprises:

interacting with a software debugger that debugs the software of the electronic system; and

interacting with a functional simulator that simulates a portion of the electronic system.

26. The machine-readable medium as recited in claim **19**, wherein the electronic system is operated in its target environment without interruption and running at its target speed during the debugging, and wherein the target environment includes real-time characteristics.

27. The machine-readable medium as recited in claim **19**, wherein the method further comprises identifying at least one fault of the electronic system, wherein the at least one fault is selected from the group consisting of specification error, design error, tool error, device driver error, timing error, manufacturing fault and environment error.

28. The machine-readable medium as recited in claim **19**, wherein the instrumentation circuitry comprises design instrumentation circuitry, wherein the method further comprises interoperating the design instrumentation circuitry with a logic analyzer, and wherein at least a portion of the debug data stems from the logic analyzer.

29. The machine-readable medium as recited in claim **19**, wherein the electronic system comprises an integrated circuit product, and wherein the method further comprises:

examining the integrated circuit product by the instrumentation circuitry; and

modifying the behavior of the integrated circuit product by the instrumentation circuitry.

30. A machine-readable medium containing instructions that when executed on a data processing system causes the system to perform a method for debugging an integrated circuit product having instrumentation circuitry included therein, wherein the integrated circuit product was designed with a high-level HDL description of the integrated circuit product, the method comprising:

activating certain aspects available for examining or modifying by the instrumentation circuitry;

determining configuration information based on the certain aspects that are activated;

configuring the instrumentation circuitry in accordance with the configuration information;

receiving debug data from the configured instrumentation circuitry operating within the integrated circuit product;

translating the debug data into HDL-related debug information;

relating the HDL-related debug information to the high-level HDL description;

retrieving circuit status information for the integrated circuit product via the instrumentation circuitry; and

displaying state information concerning the integrated circuit product based on the retrieved circuit status information.

31. The machine-readable medium as recited in claim **30**, wherein at least a portion of the HDL description of the electronic system being written in a language selected from the group consisting of HDL, Verilog HDL, C, C++ and SystemC.

32. The machine-readable medium as recited in claim **30**, wherein the HDL-related debug information is described in a high-level HDL.

33. The machine-readable medium as recited in claim **30**, wherein displaying comprises:

relating the state information to the high-level HDL description; and

displaying the high-level HDL description of the integrated circuit product with the state information related thereto, wherein the state information includes signal values for signals, and wherein relating operates to relate the signal values to HDL identifiers within the high-level HDL description that correspond to the signals.

* * * * *

CERTIFICATE OF SERVICE

I hereby certify that I electronically filed the foregoing with the Clerk of the Court for the United States Court of Appeals for the Federal Circuit by using the appellate CM/ECF system.

I certify that all participants in the case are registered CM/ECF users and that service will be accomplished by the appellate CM/ECF system.

ORRICK, HERRINGTON & SUTCLIFFE LLP

/s/ E. Joshua Rosenkranz
E. Joshua Rosenkranz

Counsel for Defendants-Appellants

CERTIFICATE OF COMPLIANCE

This brief complies with the type-volume limitation of Fed. R. App. P. 32(a)(7)(B)(i) because this brief contains 13,681 words, excluding the parts of the brief exempted by Fed. R. App. P. 32(a)(7)(B)(iii).

This brief complies with the typeface requirements of Fed. R. App. P. 32(a)(5) and the type style requirements of Fed. R. App. P. 32(a)(6) because this brief has been prepared in a proportionally spaced typeface using Microsoft Word 2010 in Century Schoolbook 14-point font.

Date: July 17, 2015

ORRICK, HERRINGTON & SUTCLIFFE LLP

/s/ E. Joshua Rosenkranz
E. Joshua Rosenkranz

Counsel for Defendants-Appellants